

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        /// </summary>
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        /// </param>
49        /// <returns>
50        /// <para>The bool</para>
51        /// </returns>
52        [MethodImpl(MethodImplOptions.AggressiveInlining)]
53        public bool IsMatched(TLink link) => _equalityComparer.Equals(_links.GetTarget(link),
54            ↪ _targetToMatch);
55    }
56 }
```

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(IList<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         /// Eatches 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(ICollection<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(ICollection<TLink> restriction, ICollection<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(IList<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(IList<TLink> restriction, IList<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(IList<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(IList<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 }

```



```

62 public delegate UInt32 UpdateCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
63     ↪ after);
64
65 public delegate UInt64 UpdateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
66     ↪ after);
67
68 public delegate Byte DeleteCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
69
70 public delegate UInt16 DeleteCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
71     ↪ after);
72
73 public delegate UInt32 DeleteCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
74     ↪ after);
75
76 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
77 public static extern void* ByteUnitedMemoryLinks_New(string path);
78
79 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
80 public static extern void* UInt16UnitedMemoryLinks_New(string path);
81
82 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
83 public static extern void* UInt32UnitedMemoryLinks_New(string path);
84
85 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
86 public static extern void* UInt64UnitedMemoryLinks_New(string path);
87
88 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
89 public static extern void ByteUnitedMemoryLinks_Drop(void* self);
90
91 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
92 public static extern void UInt16UnitedMemoryLinks_Drop(void* self);
93
94 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
95 public static extern void UInt32UnitedMemoryLinks_Drop(void* self);
96
97 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
98 public static extern void UInt64UnitedMemoryLinks_Drop(void* self);
99
100 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
101 public static extern byte ByteUnitedMemoryLinks_Create(void* self, byte* substitution,
102     ↪ nuint substitutionLength, CreateCallback_UInt8 callback);
103
104 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
105 public static extern ushort UInt16UnitedMemoryLinks_Create(void* self, ushort*
106     ↪ substitution, nuint substitutionLength, CreateCallback_UInt16 callback);
107
108 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
109 public static extern uint UInt32UnitedMemoryLinks_Create(void* self, uint* substitution,
110     ↪ nuint substitutionLength, CreateCallback_UInt32 callback);
111
112 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
113 public static extern ulong UInt64UnitedMemoryLinks_Create(void* self, ulong*
114     ↪ substitution, nuint substitutionLength, CreateCallback_UInt64 callback);
115
116 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
117 public static extern byte ByteUnitedMemoryLinks_Count(void* self, byte* restriction,
118     ↪ nuint len);
119
120 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
121 public static extern ushort UInt16UnitedMemoryLinks_Count(void* self, ushort*
122     ↪ restriction, nuint len);
123
124 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
125 public static extern uint UInt32UnitedMemoryLinks_Count(void* self, uint* restriction,
126     ↪ nuint len);
127
128 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
129 public static extern ulong UInt64UnitedMemoryLinks_Count(void* self, ulong* restriction,
130     ↪ nuint len);
131
132 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
133 public static extern byte ByteUnitedMemoryLinks_Each(void* self, byte* restriction,
134     ↪ nuint len, EachCallback_UInt8 callback);

```

```

126 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
127 public static extern ushort UInt16UnitedMemoryLinks_Each(void* self, ushort*
    ↳ restriction, nuint len, EachCallback UInt16 callback);
128
129 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
130 public static extern uint UInt32UnitedMemoryLinks_Each(void* self, uint* restriction,
    ↳ nuint len, EachCallback UInt32 callback);
131
132 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
133 public static extern ulong UInt64UnitedMemoryLinks_Each(void* self, ulong* restriction,
    ↳ nuint len, EachCallback UInt64 callback);
134
135 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
136 public static extern byte ByteUnitedMemoryLinks_Update(void* self, byte* restriction,
    ↳ nuint restrictionLength, byte* substitution, nuint substitutionLength,
    ↳ UpdateCallback UInt8 callback);
137
138 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
139 public static extern ushort UInt16UnitedMemoryLinks_Update(void* self, ushort*
    ↳ restriction, nuint restrictionLength, ushort* substitution, nuint
    ↳ substitutionLength, UpdateCallback UInt16 callback);
140
141 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
142 public static extern uint UInt32UnitedMemoryLinks_Update(void* self, uint* restriction,
    ↳ nuint restrictionLength, uint* substitution, nuint substitutionLength,
    ↳ UpdateCallback UInt32 callback);
143
144 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
145 public static extern ulong UInt64UnitedMemoryLinks_Update(void* self, ulong*
    ↳ restriction, nuint restrictionLength, ulong* substitution, nuint
    ↳ substitutionLength, UpdateCallback UInt64 callback);
146
147 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
148 public static extern byte ByteUnitedMemoryLinks_Delete(void* self, byte* restriction,
    ↳ nuint restrictionLength, DeleteCallback UInt8 callback);
149
150 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
151 public static extern ushort UInt16UnitedMemoryLinks_Delete(void* self, ushort*
    ↳ restriction, nuint len, DeleteCallback UInt16 callback);
152
153 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
154 public static extern uint UInt32UnitedMemoryLinks_Delete(void* self, uint* restriction,
    ↳ nuint len, DeleteCallback UInt32 callback);
155
156 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
157 public static extern ulong UInt64UnitedMemoryLinks_Delete(void* self, ulong*
    ↳ restriction, nuint len, DeleteCallback UInt64 callback);
158 }

```

```

159
160 public class UnitedMemoryLinks<TLink> : DisposableBase, ILinks<TLink>
161 {
162     private static readonly UncheckedConverter<byte, TLink> from_u8 =
        ↳ UncheckedConverter<byte, TLink>.Default;
163     private static readonly UncheckedConverter<ushort, TLink> from_u16 =
        ↳ UncheckedConverter<ushort, TLink>.Default;
164     private static readonly UncheckedConverter<uint, TLink> from_u32 =
        ↳ UncheckedConverter<uint, TLink>.Default;
165     private static readonly UncheckedConverter<ulong, TLink> from_u64 =
        ↳ UncheckedConverter<ulong, TLink>.Default;
166     private static readonly UncheckedConverter<TLink, ulong> from_t =
        ↳ UncheckedConverter<TLink, ulong>.Default;
167
168     public LinksConstants<TLink> Constants { get; }
169
170     private readonly unsafe void* _ptr;
171
172     public UnitedMemoryLinks(string path)
173     {
174         TLink t = default;
175         unsafe
176         {
177             _ptr = t switch
178             {
179                 byte => Methods.ByteUnitedMemoryLinks_New(path),
180                 ushort => Methods.UInt16UnitedMemoryLinks_New(path),
181                 uint => Methods.UInt32UnitedMemoryLinks_New(path),
182                 ulong => Methods.UInt64UnitedMemoryLinks_New(path),
183                 _ => throw new NotImplementedException()
184             };

```



```

185
186 // TODO: Update api
187 Constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
188 }
189 }
190
191 public TLink Count(IList<TLink> restriction)
192 {
193     unsafe
194     {
195         TLink t = default;
196         switch (t)
197         {
198             case byte:
199             {
200                 var array = stackalloc byte[restriction.Count];
201                 for (var i = 0; i < restriction.Count; i++)
202                 {
203                     array[i] = (byte)from_t.Convert(restriction[i]);
204                 }
205                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Count(_ptr, array,
206                                     ↪ (nuint)(restriction?.Count ?? 0)));
207             }
208             case ushort:
209             {
210                 var array = stackalloc ushort[restriction.Count];
211                 for (var i = 0; i < restriction.Count; i++)
212                 {
213                     array[i] = (ushort)from_t.Convert(restriction[i]);
214                 }
215                 return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Count(_ptr,
216                                     ↪ array, (nuint)(restriction?.Count ?? 0)));
217             }
218             case uint:
219             {
220                 var array = stackalloc uint[restriction.Count];
221                 for (var i = 0; i < restriction.Count; i++)
222                 {
223                     array[i] = (uint)from_t.Convert(restriction[i]);
224                 }
225                 return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Count(_ptr,
226                                     ↪ array, (nuint)(restriction?.Count ?? 0)));
227             }
228             case ulong:
229             {
230                 {
231                     var array = stackalloc UInt64[restriction.Count];
232                     for (var i = 0; i < restriction.Count; i++)
233                     {
234                         array[i] = from_t.Convert(restriction[i]);
235                     }
236                     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Count(_ptr,
237                                     ↪ array, (nuint)(restriction?.Count ?? 0)));
238                 }
239             }
240             default:
241             {
242                 throw new NotImplementedException();
243             }
244         }
245     }
246 }
247
248 public TLink Each(IList<TLink> restriction, ReadHandler<TLink> handler)
249 {
250     unsafe
251     {
252         TLink t = default;
253         switch (t)
254         {
255             case byte:
256             {
257                 Methods.EachCallback_UInt8 callback = (link) =>
258                 ↪ (byte)from_t.Convert(handler != null? handler(new
259                 ↪ Link<TLink>(from_u8.Convert(link.Index),
260                 ↪ from_u8.Convert(link.Source), from_u8.Convert(link.Target))) :
261                 ↪ Constants.Continue);
262                 var array = stackalloc byte[restriction.Count];

```

```

255     for (var i = 0; i < restriction.Count; i++)
256     {
257         array[i] = (byte)from_t.Convert(restriction[i]);
258     }
259     return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Each(_ptr, array,
        ↪ (nuint)(restriction?.Count ?? 0), callback));
260 }
261 case ushort:
262 {
263     Methods.EachCallback_UInt16 callback = (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);
264     var array = stackalloc ushort[restriction.Count];
265     for (var i = 0; i < restriction.Count; i++)
266     {
267         array[i] = (ushort)from_t.Convert(restriction[i]);
268     }
269     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Each(_ptr,
        ↪ array, (nuint)(restriction?.Count ?? 0), callback));
270 }
271 case uint:
272 {
273     Methods.EachCallback_UInt32 callback = (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);
274     var array = stackalloc uint[restriction.Count];
275     for (var i = 0; i < restriction.Count; i++)
276     {
277         array[i] = (uint)from_t.Convert(restriction[i]);
278     }
279     return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Each(_ptr,
        ↪ array, (nuint)(restriction?.Count ?? 0), callback));
280 }
281 case ulong:
282 {
283     {
284         Methods.EachCallback_UInt64 callback = (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);
285         var array = stackalloc UInt64[restriction.Count];
286         for (var i = 0; i < restriction.Count; i++)
287         {
288             array[i] = from_t.Convert(restriction[i]);
289         }
290         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Each(_ptr,
            ↪ array, (nuint)(restriction?.Count ?? 0), callback));
291     }
292 }
293 default:
294 {
295     throw new NotImplementedException();
296 }
297 }
298 }
299 }
300
301 public TLink Create(IList<TLink> substitution, WriteHandler<TLink> handler)
302 {
303     unsafe
304     {
305         TLink t = default;
306         switch (t)
307         {
308             case byte:
309             {
310                 Methods.CreateCallback_UInt8 callback = (before, after) =>
                    ↪ (byte)from_t.Convert(handler != null? handler(new
                    ↪ Link<TLink>(from_u8.Convert(before.Index),
                    ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
                    ↪ Link<TLink>(from_u8.Convert(after.Index),
                    ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
                    ↪ Constants.Continue);

```

```

311         fixed (byte* substitutionPtr = (byte[])(object)substitution)
312         {
313             return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Create(_ptr,
314                                     ↪ substitutionPtr, (nuint)(substitution?.Count ?? 0), callback));
315         }
316     case ushort:
317     {
318         Methods.CreateCallback_UInt16 callback = (before, after) =>
319             (byte)from_t.Convert(handler != null? handler(new
320                 ↪ Link<TLink>(from_u16.Convert(before.Index),
321                 ↪ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
322                 ↪ new Link<TLink>(from_u16.Convert(after.Index),
323                 ↪ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
324                 ↪ Constants.Continue);
325         fixed (ushort* substitutionPtr = (ushort[])(object)substitution)
326         {
327             return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Create(_ptr,
328                                     ↪ substitutionPtr, (nuint)(substitution?.Count ?? 0), callback));
329         }
330     }
331     case uint:
332     {
333         Methods.CreateCallback_UInt32 callback = (before, after) =>
334             (byte)from_t.Convert(handler != null? handler(new
335                 ↪ Link<TLink>(from_u32.Convert(before.Index),
336                 ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
337                 ↪ new Link<TLink>(from_u32.Convert(after.Index),
338                 ↪ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) :
339                 ↪ Constants.Continue);
340         fixed (uint* substitutionPtr = (uint[])(object)substitution)
341         {
342             return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Create(_ptr,
343                                     ↪ substitutionPtr, (nuint)(substitution?.Count ?? 0), callback));
344         }
345     }
346     case ulong:
347     {
348         Methods.CreateCallback_UInt64 callback = (before, after) =>
349             (byte)from_t.Convert(handler != null? handler(new
350                 ↪ Link<TLink>(from_u64.Convert(before.Index),
351                 ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
352                 ↪ new Link<TLink>(from_u64.Convert(after.Index),
353                 ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
354                 ↪ Constants.Continue);
355         fixed (ulong* substitutionPtr = (ulong[])(object)substitution)
356         {
357             return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Create(_ptr,
358                                     ↪ substitutionPtr, (nuint)(substitution?.Count ?? 0), callback));
359         }
360     }
361     default:
362     {
363         throw new NotImplementedException();
364     }
365 };
366
367 }
368
369 public TLink Update(IList<TLink> restriction, IList<TLink> substitution,
370     ↪ WriteHandler<TLink> handler)
371 {
372     unsafe
373     {
374         TLink t = default;
375         switch (t)
376         {
377             case byte:
378             {
379                 var restrictionArray = restriction.ToArray();
380                 var substitutionArray = substitution.ToArray();
381                 Methods.UpdateCallback_UInt8 callback = (before, after) =>
382                     (byte)from_t.Convert(handler != null? handler(new
383                         ↪ Link<TLink>(from_u8.Convert(before.Index),
384                         ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
385                         ↪ Link<TLink>(from_u8.Convert(after.Index),
386                         ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
387                         ↪ Constants.Continue);

```

```

360     fixed (byte* restrictionPointer = (byte[])(object)restrictionArray,
361           ↪ substitutionPointer = (byte[])(object)substitutionArray)
362     {
363         return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Update(_ptr,
364           ↪ restrictionPointer, (nuint)(restrictionArray?.Length ?? 0),
365           ↪ substitutionPointer, (nuint)(substitutionArray?.Length ?? 0),
366           ↪ callback));
367     }
368 }
369 case ushort:
370 {
371     var restrictionArray = restriction.ToArray();
372     var substitutionArray = substitution.ToArray();
373     Methods.UpdateCallback_UInt16 callback = (before, after) =>
374     {
375         ↪ (ushort)from_t.Convert(handler != null? handler(new
376         ↪ Link<TLink>(from_u16.Convert(before.Index),
377         ↪ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
378         ↪ new Link<TLink>(from_u16.Convert(after.Index),
379         ↪ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
380         ↪ Constants.Continue);
381     fixed (ushort* restrictionPointer = (ushort[])(object)restrictionArray,
382           ↪ substitutionPointer = (ushort[])(object)substitutionArray)
383     {
384         return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Update(_ptr,
385           ↪ restrictionPointer, (nuint)(restrictionArray?.Length ?? 0),
386           ↪ substitutionPointer, (nuint)(substitutionArray?.Length ?? 0),
387           ↪ callback));
388     }
389 }
390 case uint:
391 {
392     var restrictionArray = restriction.ToArray();
393     var substitutionArray = substitution.ToArray();
394     Methods.UpdateCallback_UInt32 callback = (before, after) =>
395     {
396         var handlerState = Constants.Continue;
397         if (handler != null)
398         {
399             handlerState = handler != null? handler(new
400             ↪ Link<TLink>(from_u32.Convert(before.Index),
401             ↪ from_u32.Convert(before.Source),
402             ↪ from_u32.Convert(before.Target)), new
403             ↪ Link<TLink>(from_u32.Convert(after.Index),
404             ↪ from_u32.Convert(after.Source),
405             ↪ from_u32.Convert(after.Target))) : Constants.Continue;
406         }
407         return (uint)from_t.Convert(handlerState);
408     };
409     fixed (uint* restrictionPointer = (uint[])(object)restrictionArray,
410           ↪ substitutionPointer = (uint[])(object)substitutionArray)
411     {
412         return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Update(_ptr,
413           ↪ restrictionPointer, (nuint)(restrictionArray?.Length ?? 0),
414           ↪ substitutionPointer, (nuint)(substitutionArray?.Length ?? 0),
415           ↪ callback));
416     }
417 }
418 case ulong:
419 {
420     Methods.UpdateCallback_UInt64 callback = (before, after) =>
421     {
422         ↪ (ulong)from_t.Convert(handler != null? handler(new
423         ↪ Link<TLink>(from_u64.Convert(before.Index),
424         ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
425         ↪ new Link<TLink>(from_u64.Convert(after.Index),
426         ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
427         ↪ Constants.Continue);
428     var restrictionArray = restriction.ToArray();
429     var substitutionArray = substitution.ToArray();
430     fixed (ulong* restrictionPointer = (ulong[])(object)restrictionArray,
431           ↪ substitutionPointer = (ulong[])(object)substitutionArray)
432     {
433         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Update(_ptr,
434           ↪ restrictionPointer, (nuint)(restrictionArray?.Length ?? 0),
435           ↪ substitutionPointer, (nuint)(substitutionArray?.Length ?? 0),
436           ↪ callback));
437     }
438 }
439 }
440 }

```

```

403         default:
404         {
405             throw new NotImplementedException();
406         }
407     }
408 }
409 }
410
411 public TLink Delete(IList<TLink> restriction, WriteHandler<TLink> handler)
412 {
413     unsafe
414     {
415         TLink t = default;
416         switch (t)
417         {
418             case byte:
419             {
420                 var restrictionArray = restriction.ToArray();
421                 Methods.DeleteCallback_UInt8 callback = (before, after) =>
422                 {
423                     (byte)from_t.Convert(handler != null? handler(new
424                     ↪ Link<TLink>(from_u8.Convert(before.Index),
425                     ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
426                     ↪ Link<TLink>(from_u8.Convert(after.Index),
427                     ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
428                     ↪ Constants.Continue);
429                 fixed (byte* restrictionPointer = (byte[])(object)restrictionArray)
430                 {
431                     return (TLink)(object)Methods.ByteUnitedMemoryLinks_Delete(_ptr,
432                     ↪ restrictionPointer, (nuint)(restrictionArray?.Length ?? 0),
433                     ↪ callback);
434                 }
435             }
436             case ushort:
437             {
438                 var restrictionArray = restriction.ToArray();
439                 Methods.DeleteCallback_UInt16 callback = (before, after) =>
440                 {
441                     (ushort)from_t.Convert(handler != null? handler(new
442                     ↪ Link<TLink>(from_u16.Convert(before.Index),
443                     ↪ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
444                     ↪ new Link<TLink>(from_u16.Convert(after.Index),
445                     ↪ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
446                     ↪ Constants.Continue);
447                 fixed (ushort* restrictionPointer = (ushort[])(object)restrictionArray)
448                 {
449                     return (TLink)(object)Methods.UInt16UnitedMemoryLinks_Delete(_ptr,
450                     ↪ restrictionPointer, (nuint)(restrictionArray?.Length ?? 0),
451                     ↪ callback);
452                 }
453             }
454             case uint:
455             {
456                 var restrictionArray = restriction.ToArray();
457                 Methods.DeleteCallback_UInt32 callback = (before, after) =>
458                 {
459                     (uint)from_t.Convert(handler != null? handler(new
460                     ↪ Link<TLink>(from_u32.Convert(before.Index),
461                     ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
462                     ↪ new Link<TLink>(from_u32.Convert(after.Index),
463                     ↪ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) :
464                     ↪ Constants.Continue);
465                 fixed (uint* restrictionPointer = (uint[])(object)restrictionArray)
466                 {
467                     return (TLink)(object)Methods.UInt32UnitedMemoryLinks_Delete(_ptr,
468                     ↪ restrictionPointer, (nuint)(restrictionArray?.Length ?? 0),
469                     ↪ callback);
470                 }
471             }
472             case ulong:
473             {
474                 var restrictionArray = restriction.ToArray();
475                 Methods.DeleteCallback_UInt64 callback = (before, after) =>
476                 {
477                     (ulong)from_t.Convert(handler != null? handler(new
478                     ↪ Link<TLink>(from_u64.Convert(before.Index),
479                     ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
480                     ↪ new Link<TLink>(from_u64.Convert(after.Index),
481                     ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
482                     ↪ Constants.Continue);
483                 fixed (ulong* restrictionPointer = (ulong[])(object)restrictionArray)
484                 {
485

```

```

451         return (TLink)(object)Methods.UInt64UnitedMemoryLinks_Delete(_ptr,
452             ↳ restrictionPointer, (nuint)(restrictionArray?.Length ?? 0),
453             ↳ callback);
454     }
455     default:
456     {
457         throw new NotImplementedException();
458     }
459 }
460 }
461
462 protected override void Dispose(bool manual, bool wasDisposed)
463 {
464     unsafe
465     {
466         if (wasDisposed && _ptr != null)
467         {
468             return;
469         }
470         TLink t = default;
471         switch (t)
472         {
473             case byte:
474                 Methods.ByteUnitedMemoryLinks_Drop(_ptr);
475                 break;
476             case ushort:
477                 Methods.UInt16UnitedMemoryLinks_Drop(_ptr);
478                 break;
479             case uint:
480                 Methods.UInt32UnitedMemoryLinks_Drop(_ptr);
481                 break;
482             case ulong:
483                 Methods.UInt64UnitedMemoryLinks_Drop(_ptr);
484                 break;
485             default:
486                 throw new NotImplementedException();
487         }
488     }
489 }
490 }
491 }

```

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 =
89             ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
90         var target =
91             ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
92         links.SearchOrDefault(source, target);
93     }
94 }
95
96 /// <summary>
97 /// <para>
98 /// Runs the random deletions using the specified links.
99 /// </para>
100 /// <para></para>
101 /// </summary>
102 /// <typeparam name="TLink">
103 /// <para>The link.</para>
104 /// <para></para>
105 /// </typeparam>
106 /// <param name="links">
107 /// <para>The links.</para>
108 /// <para></para>
109 /// </param>
110 /// <param name="amountOfDeletions">
111 /// <para>The amount of deletions.</para>
112 /// <para></para>
113 /// </param>
114 [MethodImpl(MethodImplOptions.AggressiveInlining)]
115 public static void RunRandomDeletions<TLink>(this ILinks<TLink> links, ulong
116     ↪ amountOfDeletions)
117 {
118     var random = RandomHelpers.Default;
119     var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
120     var uInt64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
121     var linksCount = addressToUInt64Converter.Convert(links.Count());
122     var min = amountOfDeletions > linksCount ? 0UL : linksCount - amountOfDeletions;
123     for (var i = 0UL; i < amountOfDeletions; i++)
124     {
125         linksCount = addressToUInt64Converter.Convert(links.Count());
126         if (linksCount <= min)
127         {
128             break;
129         }
130         var linksAddressRange = new Range<ulong>(min, linksCount);
131         var link =
132             ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
133         links.Delete(link);
134     }
135 }
136
137 /// <summary>
138 /// <para>
139 /// Deletes the links.
140 /// </para>
141 /// <para></para>
142 /// </summary>
143 /// <typeparam name="TLink">
144 /// <para>The link.</para>
145 /// <para></para>
146 /// </typeparam>
147 /// <param name="links">
148 /// <para>The links.</para>
149 /// <para></para>
150 /// </param>
151 /// <param name="linkToDelete">
152 /// <para>The link to delete.</para>
153 /// <para></para>
154 /// </param>
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 public static TLink Delete<TLink>(this ILinks<TLink> links, TLink linkToDelete,
157     ↪ WriteHandler<TLink> handler)
158 {
159     if (links.Exists(linkToDelete))
160     {
161         links.EnforceResetValues(linkToDelete, handler);
162     }
163     return links.Delete(new LinkAddress<TLink>(linkToDelete), handler);
164 }

```



```

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 =
173         ↪ Arithmetic.Decrement(i))
174     {
175         links.Delete(i);
176         if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
177         {
178             i = links.Count();
179         }
180     }
181 }
182
183 /// <summary>
184 /// <para>
185 /// Firsts the links.
186 /// </para>
187 /// <para></para>
188 /// </summary>
189 /// <typeparam name="TLink">
190 /// <para>The link.</para>
191 /// <para></para>
192 /// </typeparam>
193 /// <param name="links">
194 /// <para>The links.</para>
195 /// <para></para>
196 /// </param>
197 /// <exception cref="InvalidOperationException">
198 /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
199 /// <para></para>
200 /// </exception>
201 /// <exception cref="InvalidOperationException">
202 /// <para>В хранилище нет связей.</para>
203 /// <para></para>
204 /// </exception>
205 /// <returns>
206 /// <para>The first link.</para>
207 /// <para></para>
208 /// </returns>
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 public static TLink First<TLink>(this ILinks<TLink> links)
211 {
212     TLink firstLink = default;
213     var equalityComparer = EqualityComparer<TLink>.Default;
214     if (equalityComparer.Equals(links.Count(), default))
215     {
216         throw new InvalidOperationException("В хранилище нет связей.");
217     }
218     links.Each(links.Constants.Any, links.Constants.Any, link =>
219     {
220         firstLink = link[links.Constants.IndexPart];
221         return links.Constants.Break;
222     });
223     if (equalityComparer.Equals(firstLink, default))
224     {
225         throw new InvalidOperationException("В процессе поиска по хранилищу не было
226             ↪ найдено связей.");
227     }
228     return firstLink;
229 }
230
231 /// <summary>
232 /// <para>
233 /// Singles the or default using the specified links.
234 /// </para>
235 /// <para></para>
236 /// </summary>
237 /// <typeparam name="TLink">
238 /// <para>The link.</para>

```

```

237     /// <para></para>
238     /// </typeparam>
239     /// <param name="links">
240     /// <para>The links.</para>
241     /// <para></para>
242     /// </param>
243     /// <param name="query">
244     /// <para>The query.</para>
245     /// <para></para>
246     /// </param>
247     /// <returns>
248     /// <para>The result.</para>
249     /// <para></para>
250     /// </returns>
251     [MethodImpl(MethodImplOptions.AggressiveInlining)]
252     public static IList<TLink> SingleOrDefault<TLink>(this ILinks<TLink> links, IList<TLink>
    ↪ query)
253     {
254         IList<TLink> result = null;
255         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
    ↪ SequenceWalker
284     /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
285     /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
286     /// </remarks>
287     [MethodImpl(MethodImplOptions.AggressiveInlining)]
288     public static bool CheckPathExistance<TLink>(this ILinks<TLink> links, params TLink[]
    ↪ path)
289     {
290         var current = path[0];
291         //EnsureLinkExists(current, "path");
292         if (!links.Exists(current))
293         {
294             return false;
295         }
296         var equalityComparer = EqualityComparer<TLink>.Default;
297         var constants = links.Constants;
298         for (var i = 1; i < path.Length; i++)
299         {
300             var next = path[i];
301             var values = links.GetLink(current);
302             var source = values[constants.SourcePart];
303             var target = values[constants.TargetPart];
304             if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
    ↪ next))
305             {
306                 //throw new InvalidOperationException(string.Format("Невозможно выбрать
    ↪ путь, так как и Source и Target совпадают с элементом пути {0}.", next));
307                 return false;
308             }
309             if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
    ↪ target))

```

```

310     {
311         //throw new InvalidOperationException(string.Format("Невозможно продолжить
        ↳ путь через элемент пути {0}", next));
312         return false;
313     }
314     current = next;
315 }
316 return true;
317 }
318
319 /// <remarks>
320 /// Может потребовать дополнительного стека для PathElement's при использовании
        ↳ SequenceWalker.
321 /// </remarks>
322 [MethodImpl(MethodImplOptions.AggressiveInlining)]
323 public static TLink GetByKeys<TLink>(this ILinks<TLink> links, TLink root, params int[]
        ↳ path)
324 {
325     links.EnsureLinkExists(root, "root");
326     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
396   ↳ содержимого.</param>
397 /// <returns>Индекс начальной связи для указанной связи.</returns>
398 [MethodImpl(MethodImplOptions.AggressiveInlining)]
399 public static TLink GetIndex<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
400   ↳ link[links.Constants.IndexPart];
401
402 /// <summary>
403 /// Возвращает индекс начальной (Source) связи для указанной связи.
404 /// </summary>
405 /// <param name="links">Хранилище связей.</param>
406 /// <param name="link">Индекс связи.</param>
407 /// <returns>Индекс начальной связи для указанной связи.</returns>
408 [MethodImpl(MethodImplOptions.AggressiveInlining)]
409 public static TLink GetSource<TLink>(this ILinks<TLink> links, TLink link) =>
410   ↳ links.GetLink(link)[links.Constants.SourcePart];
411
412 /// <summary>
413 /// Возвращает индекс начальной (Source) связи для указанной связи.
414 /// </summary>
415 /// <param name="links">Хранилище связей.</param>
416 /// <param name="link">Связь представленная списком, состоящим из её адреса и
417   ↳ содержимого.</param>
418 /// <returns>Индекс начальной связи для указанной связи.</returns>
419 [MethodImpl(MethodImplOptions.AggressiveInlining)]
420 public static TLink GetSource<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
421   ↳ link[links.Constants.SourcePart];
422
423 /// <summary>
424 /// Возвращает индекс конечной (Target) связи для указанной связи.
425 /// </summary>
426 /// <param name="links">Хранилище связей.</param>
427 /// <param name="link">Индекс связи.</param>
428 /// <returns>Индекс конечной связи для указанной связи.</returns>
429 [MethodImpl(MethodImplOptions.AggressiveInlining)]
430 public static TLink GetTarget<TLink>(this ILinks<TLink> links, TLink link) =>
431   ↳ links.GetLink(link)[links.Constants.TargetPart];
432
433 /// <summary>
434 /// Возвращает индекс конечной (Target) связи для указанной связи.
435 /// </summary>
436 /// <param name="links">Хранилище связей.</param>
437 /// <param name="link">Связь представленная списком, состоящим из её адреса и
438   ↳ содержимого.</param>
439 /// <returns>Индекс конечной связи для указанной связи.</returns>
440 [MethodImpl(MethodImplOptions.AggressiveInlining)]
441 public static TLink GetTarget<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
442   ↳ link[links.Constants.TargetPart];
443
444 /// <summary>
445 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
446   ↳ (handler) для каждой подходящей связи.
447 /// </summary>
448 /// <param name="links">Хранилище связей.</param>
449 /// <param name="handler">Обработчик каждой подходящей связи.</param>
450 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
451   ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Any -
452   ↳ отсутствие ограничения, 1..∞ конкретный адрес связи.</param>
453 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
454   ↳ случае.</returns>
455 [MethodImpl(MethodImplOptions.AggressiveInlining)]
456 public static bool Each<TLink>(this ILinks<TLink> links, ReadHandler<TLink> handler,
457   ↳ params TLink[] restriction)
458   => EqualityComparer<TLink>.Default.Equals(links.Each(restriction, handler),
459     ↳ links.Constants.Continue);

```

```

446 public static bool Each<TLink>(this ILinks<TLink> links, Func<TLink, bool> handler,
447     ↳ TLink source, TLink target) => links.Each(source, target, handler);
448
449
450 /// <summary>
451 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
452     ↳ (handler) для каждой подходящей связи.
453 /// </summary>
454 /// <param name="links">Хранилище связей.</param>
455 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
456     ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
457     ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
458 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
459     ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
460     ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
461 /// <param name="handler">Обработчик каждой подходящей связи.</param>
462 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
463     ↳ случае.</returns>
464 [MethodImpl(MethodImplOptions.AggressiveInlining)]
465 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
466     ↳ Func<TLink, bool> handler)
467 {
468     var constants = links.Constants;
469     return links.Each(link => handler(link[constants.IndexPart]) ? constants.Continue :
470         ↳ constants.Break, constants.Any, source, target);
471 }
472
473 public static bool Each<TLink>(this ILinks<TLink> links, ReadHandler<TLink> handler,
474     ↳ TLink source, TLink target) => links.Each(source, target, handler);
475
476
477 /// <summary>
478 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
479     ↳ (handler) для каждой подходящей связи.
480 /// </summary>
481 /// <param name="links">Хранилище связей.</param>
482 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
483     ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
484     ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
485 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
486     ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
487     ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
488 /// <param name="handler">Обработчик каждой подходящей связи.</param>
489 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
490     ↳ случае.</returns>
491 [MethodImpl(MethodImplOptions.AggressiveInlining)]
492 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
493     ↳ ReadHandler<TLink> handler) => links.Each(handler, links.Constants.Any, source,
494     ↳ target);
495
496
497 /// <summary>
498 /// <para>
499 /// Alls the links.
500 /// </para>
501 /// <para></para>
502 /// </summary>
503 /// <typeparam name="TLink">
504 /// <para>The link.</para>
505 /// <para></para>
506 /// </typeparam>
507 /// <param name="links">
508 /// <para>The links.</para>
509 /// <para></para>
510 /// </param>
511 /// <param name="restriction">
512 /// <para>The restriction.</para>
513 /// <para></para>
514 /// </param>
515 /// <returns>
516 /// <para>A list of i list t link</para>
517 /// <para></para>
518 /// </returns>
519 [MethodImpl(MethodImplOptions.AggressiveInlining)]
520 public static IList<IList<TLink>> All<TLink>(this ILinks<TLink> links, params TLink[]
521     ↳ restriction)
522 {

```

```

504     var allLinks = new List<ILink<TLink>>();
505     var filler = new ListFiller<ILink<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>
578     ↳ restriction)
579 {
580     for (var i = 0; i < restriction.Count; i++)
581     {
582         if (!links.Exists(restriction[i]))
583         {
584             throw new ArgumentLinkDoesNotExistsException<TLink>(restriction[i],
585                 ↳ $"sequence[{i}]");
586         }
587     }
588 }
589
590 /// <summary>
591 /// <para>
592 /// Ensures the inner reference exists using the specified links.
593 /// </para>
594 /// </summary>
595 /// <typeparam name="TLink">
596 /// <para>The link.</para>
597 /// </typeparam>
598 /// <param name="links">
599 /// <para>The links.</para>
600 /// </param>
601 /// <param name="reference">
602 /// <para>The reference.</para>
603 /// </param>
604 /// <param name="argumentName">
605 /// <para>The argument name.</para>
606 /// </param>
607 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
608 /// <para></para>
609 /// </exception>
610 [MethodImpl(MethodImplOptions.AggressiveInlining)]
611 public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links, TLink
612     ↳ reference, string argumentName)
613 {
614     if (links.Constants.IsInternalReference(reference) && !links.Exists(reference))
615     {
616         throw new ArgumentLinkDoesNotExistsException<TLink>(reference, argumentName);
617     }
618 }
619
620 /// <summary>
621 /// <para>
622 /// Ensures the inner reference exists using the specified links.
623 /// </para>
624 /// </summary>
625 /// <typeparam name="TLink">
626 /// <para>The link.</para>
627 /// </typeparam>
628 /// <param name="links">
629 /// <para>The links.</para>
630 /// </param>
631 /// <param name="restriction">
632 /// <para>The restriction.</para>
633 /// </param>
634 /// <param name="argumentName">
635 /// <para>The argument name.</para>
636 /// </param>
637 [MethodImpl(MethodImplOptions.AggressiveInlining)]
638 public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links,
639     ↳ IList<TLink> restriction, string argumentName)
640 {
641     for (int i = 0; i < restriction.Count; i++)
642     {
643

```

```

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 =>
797         ↪ !links.Exists(x)));
798     if (nonExistentAddresses.Count > 0)
799     {
800         var max = nonExistentAddresses.Max();
801         max = uInt64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
802             ↪ Convert(max),
803             ↪ addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
804             ↪ imum)));
805         var createdLinks = new List<TLink>();
806         var equalityComparer = EqualityComparer<TLink>.Default;
807         TLink createdLink = creator();
808         while (!equalityComparer.Equals(createdLink, max))
809         {
810             createdLinks.Add(createdLink);
811         }
812         for (var i = 0; i < createdLinks.Count; i++)
813         {
814             if (!nonExistentAddresses.Contains(createdLinks[i]))
815             {
816                 links.Delete(createdLinks[i]);
817             }
818         }
819     }
820 }
821 #endregion
822
823 /// <param name="links">Хранилище связей.</param>
824 [MethodImpl(MethodImplOptions.AggressiveInlining)]
825 public static TLink CountUsages<TLink>(this ILinks<TLink> links, TLink link)
826 {
827     var constants = links.Constants;
828     var values = links.GetLink(link);
829     TLink usagesAsSource = links.Count(new Link<TLink>(constants.Any, link,
830         ↪ constants.Any));
831     var equalityComparer = EqualityComparer<TLink>.Default;
832     if (equalityComparer.Equals(values[constants.SourcePart], link))
833     {
834         usagesAsSource = Arithmetic<TLink>.Decrement(usagesAsSource);
835     }
836     TLink usagesAsTarget = links.Count(new Link<TLink>(constants.Any, constants.Any,
837         ↪ link));
838     if (equalityComparer.Equals(values[constants.TargetPart], link))
839     {
840         usagesAsTarget = Arithmetic<TLink>.Decrement(usagesAsTarget);
841     }
842     return Arithmetic<TLink>.Add(usagesAsSource, usagesAsTarget);
843 }
844
845 /// <param name="links">Хранилище связей.</param>
846 [MethodImpl(MethodImplOptions.AggressiveInlining)]
847 public static bool HasUsages<TLink>(this ILinks<TLink> links, TLink link) =>
848     ↪ Comparer<TLink>.Default.Compare(links.CountUsages(link), default) > 0;
849
850 /// <param name="links">Хранилище связей.</param>
851 [MethodImpl(MethodImplOptions.AggressiveInlining)]
852 public static bool Equals<TLink>(this ILinks<TLink> links, TLink link, TLink source,
853     ↪ TLink target)
854 {
855     var constants = links.Constants;
856     var values = links.GetLink(link);
857     var equalityComparer = EqualityComparer<TLink>.Default;
858     return equalityComparer.Equals(values[constants.SourcePart], source) &&
859         ↪ equalityComparer.Equals(values[constants.TargetPart], target);
860 }
861
862 /// <summary>
863 /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
864 /// </summary>
865 /// <param name="links">Хранилище связей.</param>
866 /// <param name="source">Индекс связи, которая является началом для искомой
867     ↪ связи.</param>
868 /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
869 /// <returns>Индекс искомой связи с указанными Source (началом) и Target
870     ↪ (концом).</returns>
871 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

862 public static TLink SearchOrDefault<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↪ target)
863 {
864     var constants = links.Constants;
865     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break, default);
866     links.Each(setter.SetFirstAndReturnFalse, constants.Any, source, target);
867     return setter.Result;
868 }
869
870 public static TLink CreatePoint<TLink>(this ILinks<TLink> links)
871 {
872     var constants = links.Constants;
873     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
874     links.CreatePoint(setter.SetFirstFromSecondListAndReturnTrue);
875     return setter.Result;
876 }
877
878 /// <param name="links">Хранилище связей.</param>
879 [MethodImpl(MethodImplOptions.AggressiveInlining)]
880 public static TLink CreatePoint<TLink>(this ILinks<TLink> links, WriteHandler<TLink>
    ↪ handler)
881 {
882     var constants = links.Constants;
883     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
    ↪ handler);
884     TLink link = default;
885     TLink HandlerWrapper(ICollection<TLink> before, ICollection<TLink> after)
886     {
887         link = after[constants.IndexPart];
888         return handlerState.Handler != null ? handlerState.Handler(before, after) :
    ↪ constants.Continue;
889     }
890     handlerState.Apply(links.Create(null, HandlerWrapper));
891     handlerState.Apply(links.Update(link, link, link, HandlerWrapper));
892     return handlerState.Result;
893 }
894
895 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↪ target)
896 {
897     var constants = links.Constants;
898     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
899     links.CreateAndUpdate(source, target, setter.SetFirstFromSecondListAndReturnTrue);
900     return setter.Result;
901 }
902
903
904 /// <param name="links">Хранилище связей.</param>
905 [MethodImpl(MethodImplOptions.AggressiveInlining)]
906 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↪ target, WriteHandler<TLink> handler)
907 {
908     var constants = links.Constants;
909     TLink createdLink = default;
910     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
    ↪ handler);
911     handlerState.Apply(links.Create(null, (before, after) =>
912     {
913         createdLink = links.GetIndex(after);
914         return handlerState.Handler != null ? handlerState.Handler(before, after) :
    ↪ constants.Continue;
915     }));
916     handlerState.Apply(links.Update(createdLink, source, target, handler));
917     return handlerState.Result;
918 }
919
920 /// <summary>
921 /// Обновляет связь с указанными началом (Source) и концом (Target)
922 /// на связь с указанными началом (NewSource) и концом (NewTarget).
923 /// </summary>
924 /// <param name="links">Хранилище связей.</param>
925 /// <param name="link">Индекс обновляемой связи.</param>
926 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
    ↪ выполняется обновление.</param>
927 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
    ↪ выполняется обновление.</param>
928 /// <returns>Индекс обновлённой связи.</returns>
929 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

930 public static TLink Update<TLink>(this ILinks<TLink> links, TLink link, TLink newSource,
    ↳ TLink newTarget) => links.Update(new LinkAddress<TLink>(link), new Link<TLink>(link,
    ↳ newSource, newTarget));
931
932 public static TLink Update<TLink>(this ILinks<TLink> links, params TLink[] restriction)
    ↳ => links.Update(restriction, null);
933
934 public static TLink Update<TLink>(this ILinks<TLink> links, WriteHandler<TLink> handler,
    ↳ params TLink[] restriction) => links.Update(restriction, handler);
935
936 public static TLink Update<TLink>(this ILinks<TLink> links, IList<TLink> restriction)
937 {
938     var constants = links.Constants;
939     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
940     links.Update(restriction, setter.SetFirstFromSecondListAndReturnTrue);
941     return setter.Result;
942 }
943
944
945 /// <summary>
946 /// Обновляет связь с указанными началом (Source) и концом (Target)
947 /// на связь с указанными началом (NewSource) и концом (NewTarget).
948 /// </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
1015   ↳ с указанными Source (началом) и Target (концом).
1016 /// </summary>
1017 /// <param name="links">Хранилище связей.</param>
1018 /// <param name="source">Индекс связи, которая является началом на создаваемой
1019   ↳ связи.</param>
1020 /// <param name="target">Индекс связи, которая является концом для создаваемой
1021   ↳ связи.</param>
1022 /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
1023 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1024 public static TLink GetOrCreate<TLink>(this ILinks<TLink> links, TLink source, TLink
1025   ↳ target)
1026 {
1027     var link = links.SearchOrDefault(source, target);
1028     if (EqualityComparer<TLink>.Default.Equals(link, default))
1029     {
1030         link = links.CreateAndUpdate(source, target);
1031     }
1032     return link;
1033 }
1034
1035 public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
1036   ↳ TLink target, TLink newSource, TLink newTarget)
1037 {
1038     var constants = links.Constants;
1039     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
1040     links.UpdateOrCreateOrGet(source, target, newSource, newTarget,
1041   ↳ setter.SetFirstFromSecondListAndReturnTrue);
1042     return setter.Result;
1043 }
1044
1045 /// <summary>
1046 /// Обновляет связь с указанными началом (Source) и концом (Target)
1047   ↳ на связь с указанными началом (NewSource) и концом (NewTarget).
1048 /// </summary>
1049 /// <param name="links">Хранилище связей.</param>
1050 /// <param name="source">Индекс связи, которая является началом обновляемой
1051   ↳ связи.</param>
1052 /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
1053 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
1054   ↳ выполняется обновление.</param>
1055 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
1056   ↳ выполняется обновление.</param>
1057 /// <returns>Индекс обновлённой связи.</returns>
1058 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1059 public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
1060   ↳ TLink target, TLink newSource, TLink newTarget, WriteHandler<TLink> handler)
1061 {
1062     var equalityComparer = EqualityComparer<TLink>.Default;
1063     var link = links.SearchOrDefault(source, target);
1064     if (equalityComparer.Equals(link, default))
1065     {
1066         return links.CreateAndUpdate(newSource, newTarget, handler);
1067     }
1068     if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
1069   ↳ target))
1070     {
1071         var linkStruct = new Link<TLink>(link, source, target);
1072         return link;
1073     }

```

```

1063         return links.Update(link, newSource, newTarget, handler);
1064     }
1065
1066     /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1067     /// <param name="links">Хранилище связей.</param>
1068     /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
1069     /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
1070     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1071     public static TLink DeleteIfExists<TLink>(this ILinks<TLink> links, TLink source, TLink
        ↪ target)
1072     {
1073         var link = links.SearchOrDefault(source, target);
1074         if (!EqualityComparer<TLink>.Default.Equals(link, default))
1075         {
1076             links.Delete(link);
1077             return link;
1078         }
1079         return default;
1080     }
1081
1082     /// <summary>Удаляет несколько связей.</summary>
1083     /// <param name="links">Хранилище связей.</param>
1084     /// <param name="deletedLinks">Список адресов связей к удалению.</param>
1085     [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) =>
        ↪ links.DeleteAllUsages(linkIndex, null);
1095
1096     /// <remarks>Before execution of this method ensure that deleted link is detached (all
        ↪ values - source and target are reset to null) or it might enter into infinite
        ↪ recursion.</remarks>
1097     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1098     public static TLink DeleteAllUsages<TLink>(this ILinks<TLink> links, TLink linkIndex,
        ↪ WriteHandler<TLink> handler)
1099     {
1100         var constants = links.Constants;
1101         var any = constants.Any;
1102         var equalityComparer = EqualityComparer<TLink>.Default;
1103         var usagesAsSourceQuery = new Link<TLink>(any, linkIndex, any);
1104         var usagesAsTargetQuery = new Link<TLink>(any, any, linkIndex);
1105         var usages = new List<ILink<TLink>>();
1106         var usagesFiller = new ListFiller<ILink<TLink>, TLink>(usages, constants.Continue);
1107         links.Each(usagesFiller.AddAndReturnConstant, usagesAsSourceQuery);
1108         links.Each(usagesFiller.AddAndReturnConstant, usagesAsTargetQuery);
1109         WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
            ↪ handler);
1110         foreach (var usage in usages)
1111         {
1112             if (equalityComparer.Equals(links.GetIndex(usage), linkIndex) ||
                ↪ !links.Exists(links.GetIndex(usage)))
1113             {
1114                 continue;
1115             }
1116             handlerState.Apply(links.Delete(links.GetIndex(usage), handlerState.Handler));
1117         }
1118         return handlerState.Result;
1119     }
1120
1121     /// <summary>
1122     /// <para>
1123     /// Deletes the by query using the specified links.
1124     /// </para>
1125     /// <para></para>
1126     /// </summary>
1127     /// <typeparam name="TLink">
1128     /// <para>The link.</para>
1129     /// <para></para>
1130     /// </typeparam>
1131     /// <param name="links">
1132     /// <para>The links.</para>
1133     /// <para></para>

```

```

1134     /// </param>
1135     /// <param name="query">
1136     /// <para>The query.</para>
1137     /// <para></para>
1138     /// </param>
1139     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1140     public static void DeleteByQuery<TLink>(this ILinks<TLink> links, Link<TLink> query)
1141     {
1142         var queryResult = new List<TLink>();
1143         var queryResultFiller = new ListFiller<TLink, TLink>(queryResult,
1144             ↪ links.Constants.Continue);
1145         links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
1146         foreach (var link in queryResult)
1147         {
1148             links.Delete(link);
1149         }
1150     }
1151     // TODO: Move to Platform.Data
1152     /// <summary>
1153     /// <para>
1154     /// Determines whether are values reset.
1155     /// </para>
1156     /// <para></para>
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     public static void ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex) =>
1190     ↪ links.ResetValues(linkIndex, null);
1191     // TODO: Create a universal version of this method in Platform.Data (with using of for
1192     ↪ loop)
1193     /// <summary>
1194     /// <para>
1195     /// Resets the values using the specified links.
1196     /// </para>
1197     /// <para></para>
1198     /// </summary>
1199     /// <typeparam name="TLink">
1200     /// <para>The link.</para>
1201     /// <para></para>
1202     /// </typeparam>
1203     /// <param name="links">
1204     /// <para>The links.</para>
1205     /// <para></para>
1206     /// </param>
1207     /// <param name="linkIndex">
1208     /// <para>The link index.</para>

```

```

1209     /// <para></para>
1210     /// </param>
1211     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1212     public static TLink ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex,
1213     ↪ WriteHandler<TLink> handler)
1214     {
1215         var nullConstant = links.Constants.Null;
1216         var updateRequest = new Link<TLink>(linkIndex, nullConstant, nullConstant);
1217         return links.Update(updateRequest, handler);
1218     }
1219
1220     public static void EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1221     ↪ => links.EnforceResetValues(linkIndex, null);
1222
1223     // TODO: Create a universal version of this method in Platform.Data (with using of for
1224     ↪ loop)
1225     /// <summary>
1226     /// <para>
1227     /// Enforces the reset values using the specified links.
1228     /// </para>
1229     /// </summary>
1230     /// <typeparam name="TLink">
1231     /// <para>The link.</para>
1232     /// </typeparam>
1233     /// <param name="links">
1234     /// <para>The links.</para>
1235     /// </param>
1236     /// <param name="linkIndex">
1237     /// <para>The link index.</para>
1238     /// </param>
1239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1240     public static TLink EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex,
1241     ↪ WriteHandler<TLink> handler)
1242     {
1243         if (!links.AreValuesReset(linkIndex))
1244         {
1245             return links.ResetValues(linkIndex, handler);
1246         }
1247         return links.Constants.Continue;
1248     }
1249
1250     public static void MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1251     ↪ 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
1255     ↪ or deleted.
1256     /// </summary>
1257     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1258     public static TLink MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1259     ↪ TLink newLinkIndex, WriteHandler<TLink> handler)
1260     {
1261         var equalityComparer = EqualityComparer<TLink>.Default;
1262         if (equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1263         {
1264             return newLinkIndex;
1265         }
1266         var constants = links.Constants;
1267         var usagesAsSource = links.All(new Link<TLink>(constants.Any, oldLinkIndex,
1268         ↪ constants.Any));
1269         WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
1270         ↪ handler);
1271         for (var i = 0; i < usagesAsSource.Count; i++)
1272         {
1273             var usageAsSource = usagesAsSource[i];
1274             if (equalityComparer.Equals(usageAsSource[constants.IndexPart], oldLinkIndex))
1275             {
1276                 continue;
1277             }
1278             var restriction = new LinkAddress<TLink>(usageAsSource[constants.IndexPart]);
1279             var substitution = new Link<TLink>(newLinkIndex,
1280             ↪ usageAsSource[constants.TargetPart]);

```



```

1276         handlerState.Apply(links.Update(restriction, substitution,
1277             ↪ handlerState.Handler));
1278     }
1279     var usagesAsTarget = links.All(new Link<TLink>(constants.Any, constants.Any,
1280         ↪ oldLinkIndex));
1281     for (var i = 0; i < usagesAsTarget.Count; i++)
1282     {
1283         var usageAsTarget = usagesAsTarget[i];
1284         if (equalityComparer.Equals(usageAsTarget[constants.IndexPart], oldLinkIndex))
1285         {
1286             continue;
1287         }
1288         var restriction = links.GetLink(usageAsTarget[constants.IndexPart]);
1289         var substitution = new Link<TLink>(usageAsTarget[constants.TargetPart],
1290             ↪ newLinkIndex);
1291         handlerState.Apply(links.Update(restriction, substitution,
1292             ↪ handlerState.Handler));
1293     }
1294     return handlerState.Result;
1295 }
1296
1297 public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1298     ↪ TLink newLinkIndex)
1299 {
1300     var equalityComparer = EqualityComparer<TLink>.Default;
1301     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1302     {
1303         links.MergeUsages(oldLinkIndex, newLinkIndex);
1304         links.Delete(oldLinkIndex);
1305     }
1306     return newLinkIndex;
1307 }
1308
1309 /// <summary>
1310 /// Replace one link with another (replaced link is deleted, children are updated or
1311 /// ↪ deleted).
1312 /// </summary>
1313 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1314 public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1315     ↪ TLink newLinkIndex, WriteHandler<TLink> handler)
1316 {
1317     var equalityComparer = EqualityComparer<TLink>.Default;
1318     var constants = links.Constants;
1319     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
1320         ↪ handler);
1321     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1322     {
1323         handlerState.Apply(links.MergeUsages(oldLinkIndex, newLinkIndex,
1324             ↪ handlerState.Handler));
1325         handlerState.Apply(links.Delete(oldLinkIndex, handlerState.Handler));
1326     }
1327     return handlerState.Result;
1328 }
1329
1330 /// <summary>
1331 /// <para>
1332 /// Decorates the with automatic uniqueness and usages resolution using the specified
1333 /// ↪ links.
1334 /// </para>
1335 /// <para></para>
1336 /// </summary>
1337 /// <typeparam name="TLink">
1338 /// <para>The link.</para>
1339 /// <para></para>
1340 /// </typeparam>
1341 /// <param name="links">
1342 /// <para>The links.</para>
1343 /// <para></para>
1344 /// </param>
1345 /// <returns>
1346 /// <para>The links.</para>
1347 /// <para></para>
1348 /// </returns>
1349 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1350 public static ILinks<TLink>
1351     ↪ DecorateWithAutomaticUniquenessAndUsagesResolution<TLink>(this ILinks<TLink> links)
1352 {
1353     links = new LinksCascadeUsagesResolver<TLink>(links);

```

```

1343         links = new NonNullContentsLinkDeletionResolver<TLink>(links);
1344         links = new LinksCascadeUniquenessAndUsagesResolver<TLink>(links);
1345         return links;
1346     }
1347
1348     /// <summary>
1349     /// <para>
1350     /// Formats the links.
1351     /// </para>
1352     /// <para></para>
1353     /// </summary>
1354     /// <typeparam name="TLink">
1355     /// <para>The link.</para>
1356     /// <para></para>
1357     /// </typeparam>
1358     /// <param name="links">
1359     /// <para>The links.</para>
1360     /// <para></para>
1361     /// </param>
1362     /// <param name="link">
1363     /// <para>The link.</para>
1364     /// <para></para>
1365     /// </param>
1366     /// <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
1378     /// <summary>
1379     /// <para>
1380     /// Formats the links.
1381     /// </para>
1382     /// <para></para>
1383     /// </summary>
1384     /// <typeparam name="TLink">
1385     /// <para>The link.</para>
1386     /// <para></para>
1387     /// </typeparam>
1388     /// <param name="links">
1389     /// <para>The links.</para>
1390     /// <para></para>
1391     /// </param>
1392     /// <param name="link">
1393     /// <para>The link.</para>
1394     /// <para></para>
1395     /// </param>
1396     /// <returns>
1397     /// <para>The string</para>
1398     /// <para></para>
1399     /// </returns>
1400     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1401     public static string Format<TLink>(this ILinks<TLink> links, TLink link) =>
1402         ↪ links.Format(links.GetLink(link));
1403 }

```

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>

```

```

14     {
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
40         /// <summary>
41         /// <para>
42         /// The source.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         public readonly TLink Source;
47
48         /// <summary>
49         /// <para>
50         /// The target.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         public readonly TLink Target;
55
56         /// <summary>
57         /// <para>
58         /// Initializes a new <see cref="Link"/> instance.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         /// <param name="values">
63         /// <para>A values.</para>
64         /// <para></para>
65         /// </param>
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         public Link(params TLink[] values) => SetValues(values, out Index, out Source, out
68             ↳ Target);
69
70         /// <summary>
71         /// <para>
72         /// Initializes a new <see cref="Link"/> instance.
73         /// </para>
74         /// <para></para>
75         /// </summary>
76         /// <param name="values">

```

```

72     /// <para>A values.</para>
73     /// <para></para>
74     /// </param>
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     public Link(IList<TLink> values) => SetValues(values, out Index, out Source, out Target);
77
78     /// <summary>
79     /// <para>
80     /// Initializes a new <see cref="Link"/> instance.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="other">
85     /// <para>A other.</para>
86     /// <para></para>
87     /// </param>
88     /// <exception cref="NotSupportedException">
89     /// <para></para>
90     /// <para></para>
91     /// </exception>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public Link(object other)
94     {
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
121     ↪ Target);
122
123     /// <summary>
124     /// <para>
125     /// Initializes a new <see cref="Link"/> instance.
126     /// </para>
127     /// <para></para>
128     /// </summary>
129     /// <param name="index">
130     /// <para>A index.</para>
131     /// <para></para>
132     /// </param>
133     /// <param name="source">
134     /// <para>A source.</para>
135     /// <para></para>
136     /// </param>
137     /// <param name="target">
138     /// <para>A target.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public Link(TLink index, TLink source, TLink target)
143     {
144         Index = index;
145         Source = source;
146         Target = target;
147     }
148     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

148 private static void SetValues(ref Link<TLink> other, out TLink index, out TLink source,
149     ↪ out TLink target)
150 {
151     index = other.Index;
152     source = other.Source;
153     target = other.Target;
154 }
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 private static void SetValues(ICollection<TLink> values, out TLink index, out TLink source,
157     ↪ out TLink target)
158 {
159     if (values == null)
160     {
161         index = default;
162         source = default;
163         target = default;
164         return;
165     }
166     switch (values.Count)
167     {
168         case 3:
169             index = values[0];
170             source = values[1];
171             target = values[2];
172             break;
173         case 2:
174             index = values[0];
175             source = values[1];
176             target = default;
177             break;
178         case 1:
179             index = values[0];
180             source = default;
181             target = default;
182             break;
183         default:
184             index = default;
185             source = default;
186             target = default;
187             break;
188     }
189 }
190
191 /// <summary>
192 /// <para>
193 /// Gets the hash code.
194 /// </para>
195 /// </summary>
196 /// <returns>
197 /// <para>The int</para>
198 /// <para></para>
199 /// </returns>
200 [MethodImpl(MethodImplOptions.AggressiveInlining)]
201 public override int GetHashCode() => (Index, Source, Target).GetHashCode();
202
203 /// <summary>
204 /// <para>
205 /// Determines whether this instance is null.
206 /// </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 /// </summary>
222 /// <param name="other">
223 /// <para>The other.</para>
224

```

```

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> &&
        ↪ Equals((Link<TLink>)other);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance equals.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="other">
241     /// <para>The other.</para>
242     /// <para></para>
243     /// </param>
244     /// <returns>
245     /// <para>The bool</para>
246     /// <para></para>
247     /// </returns>
248     [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 [MethodImpl(MethodImplOptions.AggressiveInlining)]
302 public static implicit operator Link<TLink>(TLink[] linkArray) => new
303     ↳ 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),
            ↳ nameof(index));
356         if (index == _constants.IndexPart)
357         {
358             return Index;
359         }
360         if (index == _constants.SourcePart)
361         {
362             return Source;
363         }
364         if (index == _constants.TargetPart)
365         {
366             return Target;
367         }
368         throw new NotSupportedException(); // Impossible path due to
            ↳ Ensure.ArgumentInRange
369     }
370     [MethodImpl(MethodImplOptions.AggressiveInlining)]
371     set => throw new NotSupportedException();
372 }
373
374 /// <summary>
375 /// <para>

```

```

376     /// Gets the enumerator.
377     /// </para>
378     /// <para></para>
379     /// </summary>
380     /// <returns>
381     /// <para>The enumerator</para>
382     /// <para></para>
383     /// </returns>
384     [MethodImpl(MethodImplOptions.AggressiveInlining)]
385     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
386
387     /// <summary>
388     /// <para>
389     /// Gets the enumerator.
390     /// </para>
391     /// <para></para>
392     /// </summary>
393     /// <returns>
394     /// <para>An enumerator of t link</para>
395     /// <para></para>
396     /// </returns>
397     [MethodImpl(MethodImplOptions.AggressiveInlining)]
398     public IEnumerator<TLink> GetEnumerator()
399     {
400         yield return Index;
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>
479 /// Determines whether this instance remove.
480 /// </para>
481 /// <para></para>
482 /// </summary>
483 /// <param name="item">
484 /// <para>The item.</para>
485 /// <para></para>
486 /// </param>
487 /// <returns>
488 /// <para>The bool</para>
489 /// <para></para>
490 /// </returns>
491 [MethodImpl(MethodImplOptions.AggressiveInlining)]
492 public bool Remove(TLink item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
493
494 /// <summary>
495 /// <para>
496 /// Indexes the of using the specified item.
497 /// </para>
498 /// <para></para>
499 /// </summary>
500 /// <param name="item">
501 /// <para>The item.</para>
502 /// <para></para>
503 /// </param>
504 /// <returns>
505 /// <para>The int</para>
506 /// <para></para>
507 /// </returns>
508 [MethodImpl(MethodImplOptions.AggressiveInlining)]
509 public int IndexOf(TLink item)
510 {
511     if (_equalityComparer.Equals(Index, item))
512     {
513         return _constants.IndexPart;
514     }
515     if (_equalityComparer.Equals(Source, item))
516     {
517         return _constants.SourcePart;
518     }
519     if (_equalityComparer.Equals(Target, item))
520     {
521         return _constants.TargetPart;
522     }
523     return -1;
524 }
525
526 /// <summary>
527 /// <para>
528 /// Inserts the index.
529 /// </para>
530 /// <para></para>
531 /// </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     #endregion
562 }
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>

```

```

42     /// <typeparam name="TLink">
43     /// <para>The link.</para>
44     /// <para></para>
45     /// </typeparam>
46     /// <param name="link">
47     /// <para>The link.</para>
48     /// <para></para>
49     /// </param>
50     /// <returns>
51     /// <para>The bool</para>
52     /// <para></para>
53     /// </returns>
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     public static bool IsPartialPoint<TLink>(this Link<TLink> link) =>
56         ↪ Point<TLink>.IsPartialPoint(link);
57 }

```

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.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     SizedAndThreadedAVLBalancedTree = 3
41 }
42 }

```

1.31 ./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Unsafe;
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     /// The links header.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct LinksHeader<TLink> : IEquatable<LinksHeader<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<LinksHeader<TLink>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The allocated links.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLink AllocatedLinks;
36
37         /// <summary>
38         /// <para>
39         /// The reserved links.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink ReservedLinks;
44
45         /// <summary>
46         /// <para>
47         /// The free links.

```

```

45     /// </para>
46     /// <para></para>
47     /// </summary>
48     public TLink FreeLinks;
49     /// <summary>
50     /// <para>
51     /// The first free link.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     public TLink FirstFreeLink;
56     /// <summary>
57     /// <para>
58     /// The root as source.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLink RootAsSource;
63     /// <summary>
64     /// <para>
65     /// The root as target.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public TLink RootAsTarget;
70     /// <summary>
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,
139     ↪ FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();
140
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static bool operator ==(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
143     ↪ left.Equals(right);
144
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static bool operator !=(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
147     ↪ !(left == right);
148 }
149 }

```

1.32 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethods

```

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 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>

```



```

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

```

```

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     /// <summary>
212     /// <para>
213     /// Gets the link index part reference 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>A ref raw link index part of t link</para>
223     /// <para></para>
224     /// </returns>
225     [MethodImpl(MethodImplOptions.AggressiveInlining)]
226     protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
        ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));

228     /// <summary>
229     /// <para>
230     /// Gets the link values using the specified link index.
231     /// </para>
232     /// <para></para>
233     /// </summary>
234     /// <param name="linkIndex">
235     /// <para>The link index.</para>
236     /// <para></para>
237     /// </param>
238     /// <returns>
239     /// <para>A list of t link</para>
240     /// <para></para>
241     /// </returns>
242     [MethodImpl(MethodImplOptions.AggressiveInlining)]
243     protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
244     {
245         ref var link = ref GetLinkDataPartReference(linkIndex);
246         return new Link<TLink>(linkIndex, link.Source, link.Target);
247     }

249     /// <summary>
250     /// <para>
251     /// Determines whether this instance first is to the left of second.
252     /// </para>
253     /// <para></para>
254     /// </summary>
255     /// <param name="first">
256     /// <para>The first.</para>
257     /// <para></para>
258     /// </param>
259     /// <param name="second">
260     /// <para>The second.</para>
261     /// <para></para>
262     /// </param>
263     /// <returns>
264     /// <para>The bool</para>
265     /// <para></para>
266     /// </returns>
267     [MethodImpl(MethodImplOptions.AggressiveInlining)]
268     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
269     {
270         ref var firstLink = ref GetLinkDataPartReference(first);
271         ref var secondLink = ref GetLinkDataPartReference(second);

```

```

273         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
274             ↪ secondLink.Source, secondLink.Target);
275     }
276     /// <summary>
277     /// <para>
278     /// Determines whether this instance first is to the right of second.
279     /// </para>
280     /// <para></para>
281     /// </summary>
282     /// <param name="first">
283     /// <para>The first.</para>
284     /// <para></para>
285     /// </param>
286     /// <param name="second">
287     /// <para>The second.</para>
288     /// <para></para>
289     /// </param>
290     /// <returns>
291     /// <para>The bool</para>
292     /// <para></para>
293     /// </returns>
294     [MethodImpl(MethodImplOptions.AggressiveInlining)]
295     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
296     {
297         ref var firstLink = ref GetLinkDataPartReference(first);
298         ref var secondLink = ref GetLinkDataPartReference(second);
299         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
300             ↪ secondLink.Source, secondLink.Target);
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
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     {

```

```

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)) //
354             ↪ node.Key < root.Key
355         {
356             root = GetLeftOrDefault(root);
357         }
358         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
359             ↪ node.Key > root.Key
360         {
361             root = GetRightOrDefault(root);
362         }
363         else // node.Key == root.Key
364         {
365             return root;
366         }
367     }
368     return Zero;
369 }
370
371 // TODO: Return indices range instead of references count
372 /// <summary>
373 /// <para>
374 /// Counts the usages using the specified link.
375 /// </para>
376 /// <para></para>
377 /// </summary>
378 /// <param name="link">
379 /// <para>The link.</para>
380 /// </param>
381 /// <returns>
382 /// <para>The link</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,
        ↳ 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 }

```

1.33 ./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>

```

```

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

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

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

211     /// <summary>
212

```

```

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,
    ↪ secondLink.Source, secondLink.Target);
274 }
275
276 /// <summary>
277 /// <para>
278 /// Determines whether this instance first is to the right of second.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 /// <param name="first">
283 /// <para>The first.</para>
284 /// <para></para>
285 /// </param>
286 /// <param name="second">
287 /// <para>The second.</para>

```

```

288     /// <para></para>
289     /// </param>
290     /// <returns>
291     /// <para>The bool</para>
292     /// <para></para>
293     /// </returns>
294     [MethodImpl(MethodImplOptions.AggressiveInlining)]
295     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
296     {
297         ref var firstLink = ref GetLinkDataPartReference(first);
298         ref var secondLink = ref GetLinkDataPartReference(second);
299         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
300             ↪ secondLink.Source, secondLink.Target);
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
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         }
365     }

```

```

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,
    ↳ 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">
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 ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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
61        ↪ GetLinkIndexPartReference(node).LeftAsSource;
62
63        /// <summary>
64        /// <para>
65        /// Gets the right reference using the specified node.
66        /// </para>
67        /// <para></para>
68        /// </summary>
69        /// <param name="node">
70        /// <para>The node.</para>
71        /// <para></para>
72        /// </param>
73        /// <returns>
74        /// <para>The ref link</para>
75        /// <para></para>
76        /// </returns>
77        [MethodImpl(MethodImplOptions.AggressiveInlining)]
78        protected override ref TLink GetRightReference(TLink node) => ref
79        ↪ GetLinkIndexPartReference(node).RightAsSource;
80
81        /// <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) =>
91         ↪ GetLinkIndexPartReference(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) =>
109        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(node).RightAsSource = right;
146
147    /// <summary>
148    /// <para>
149    /// Gets the size using the specified node.
150    /// </para>
151    /// <para></para>
152    /// </summary>
153    /// <param name="node">

```

```

150    /// <para>The node.</para>
151    /// <para></para>
152    /// </param>
153    /// <returns>
154    /// <para>The link</para>
155    /// <para></para>
156    /// </returns>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override TLink GetSize(TLink node) =>
159        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(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() => GetHeaderReference().RootAsSource;
191
192    /// <summary>
193    /// <para>
194    /// Gets the base part value using the specified link.
195    /// </para>
196    /// <para></para>
197    /// </summary>
198    /// <param name="link">
199    /// <para>The link.</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 link) =>
208        ↪ GetLinkDataPartReference(link).Source;
209
210    /// <summary>
211    /// <para>
212    /// Determines whether this instance first is to the left of second.
213    /// </para>
214    /// <para></para>
215    /// </summary>
216    /// <param name="firstSource">
217    /// <para>The first source.</para>
218    /// <para></para>
219    /// </param>
220    /// <param name="firstTarget">
221    /// <para>The first target.</para>
222    /// <para></para>
223    /// </param>
224    /// <param name="secondSource">
225    /// <para>The second source.</para>
226    /// <para></para>
227    /// </param>

```



```

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

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

264     /// <summary>
265     /// <para>
266     /// <para>Clears the node using the specified node.
267     /// </para>
268     /// </para>
269     /// </summary>
270     /// <param name="node">
271     /// <para>The node.</para>
272     /// </para>
273     /// </param>
274     [MethodImpl(MethodImplOptions.AggressiveInlining)]
275     protected override void ClearNode(TLink node)
276     {
277         ref var link = ref GetLinkIndexPartReference(node);
278         link.LeftAsSource = Zero;
279         link.RightAsSource = Zero;
280         link.SizeAsSource = Zero;
281     }
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     /// <para>Represents the external links sources size balanced tree methods.
10    /// </para>
11    /// </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) =>
91         ↪ GetLinkIndexPartReference(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) =>
109        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(node).RightAsSource = right;
146
147    /// <summary>
148    /// <para>
149    /// Gets the size using the specified node.
150    /// </para>
151    /// <para></para>
152    /// </summary>
153    /// <param name="node">
154    /// <para>The node.</para>
155    /// <para></para>
156    /// </param>
157    /// <returns>
158    /// <para>The link</para>
159    /// <para></para>
160    /// </returns>
161    [MethodImpl(MethodImplOptions.AggressiveInlining)]
162    protected override TLink GetSize(TLink node) =>
163        ↪ GetLinkIndexPartReference(node).SizeAsSource;

```

```

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>
169     /// <param name="size">
170     /// <para>The size.</para>
171     /// <para></para>
172     /// </param>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override void SetSize(TLink node, TLink size) =>
175     ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
176
177     /// <summary>
178     /// <para>
179     /// Gets the tree root.
180     /// </para>
181     /// <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)]

```

```

234     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
    ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
235
236     /// <summary>
237     /// <para>
238     /// Determines whether this instance first is to the right of second.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="firstSource">
243     /// <para>The first source.</para>
244     /// <para></para>
245     /// </param>
246     /// <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 }

```

1.36 ./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>

```

```

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 ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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) =>
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   /// <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).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       ↪ GetLinkIndexPartReference(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) =>
162       ↪ GetLinkIndexPartReference(node).SizeAsTarget;
163
164   /// <summary>
165   /// <para>
166   /// Sets the size using the specified node.
167   /// </para>
168   /// <para></para>
169   /// </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).SizeAsTarget = 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().RootAsTarget;
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).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 FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
237     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
238     ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>

```



```

240     /// <para></para>
241     /// </summary>
242     /// <param name="firstSource">
243     /// <para>The first source.</para>
244     /// <para></para>
245     /// </param>
246     /// <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(firstTarget, secondTarget) ||
        ↪ (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> :
        ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="ExternalLinksTargetsSizeBalancedTreeMethods"/> 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 ExternalLinksTargetsSizeBalancedTreeMethods(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) =>
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     /// <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).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         ↪ GetLinkIndexPartReference(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) =>
162         ↪ GetLinkIndexPartReference(node).SizeAsTarget;
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>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```
protected override void SetSize(TLink node, TLink size) =>
    ↳ GetLinkIndexPartReference(node).SizeAsTarget = size;

/// <summary>
/// <para>
/// Gets the tree root.
/// </para>
/// <para></para>
/// </summary>
/// <returns>
/// <para>The link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;

/// <summary>
/// <para>
/// Gets the base part value using the specified link.
/// </para>
/// <para></para>
/// </summary>
/// <param name="link">
/// <para>The link.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override TLink GetBasePartValue(TLink link) =>
    ↳ GetLinkDataPartReference(link).Target;

/// <summary>
/// <para>
/// Determines whether this instance first is to the left of second.
/// </para>
/// <para></para>
/// </summary>
/// <param name="firstSource">
/// <para>The first source.</para>
/// <para></para>
/// </param>
/// <param name="firstTarget">
/// <para>The first target.</para>
/// <para></para>
/// </param>
/// <param name="secondSource">
/// <para>The second source.</para>
/// <para></para>
/// </param>
/// <param name="secondTarget">
/// <para>The second target.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The bool</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↳ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
    ↳ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));

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

```

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(firstTarget, secondTarget) ||
        ↪ (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 }

```

1.38 ./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> :
        ↪ RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
23     {
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
65     ↪ cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="constants">
70     /// <para>A constants.</para>
71     /// <para></para>
72     /// </param>
73     /// <param name="linksDataParts">
74     /// <para>A links data parts.</para>
75     /// <para></para>
76     /// </param>
77     /// <param name="linksIndexParts">
78     /// <para>A links index parts.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="header">
82     /// <para>A header.</para>
83     /// <para></para>
84     /// </param>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
87     ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
88     {
89         LinksDataParts = linksDataParts;
90         LinksIndexParts = linksIndexParts;
91         Header = header;
92         Break = constants.Break;
93         Continue = constants.Continue;
94     }
95
96     /// <summary>
97     /// <para>
98     /// Gets the tree root using the specified link.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    /// <param name="link">
103    /// <para>The link.</para>
104    /// <para></para>
105    /// </param>
106    /// <returns>
107    /// <para>The link</para>
108    /// <para></para>
109    /// </returns>
110    [MethodImpl(MethodImplOptions.AggressiveInlining)]
111    protected abstract TLink GetTreeRoot(TLink link);
112
113    /// <summary>
114    /// <para>
115    /// 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 FirstIsToTheLeftOfSecond(TLink first, TLink second) =>
199         ↳ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance first is to the right of second.
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 FirstIsToTheRightOfSecond(TLink first, TLink second) =>
221         ↳ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
222
223     /// <summary>
224     /// <para>
225     /// Gets the link values using the specified link index.
226     /// </para>
227     /// <para></para>
228     /// </summary>
229     /// <param name="linkIndex">
230     /// <para>The link index.</para>
231     /// <para></para>
232     /// </param>
233     /// <returns>
234     /// <para>A list of t link</para>
235     /// <para></para>
236     /// </returns>
237     [MethodImpl(MethodImplOptions.AggressiveInlining)]
238     protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
239     {
240         ref var link = ref GetLinkDataPartReference(linkIndex);
241         return new Link<TLink>(linkIndex, link.Source, link.Target);
242     }
243
244     /// <summary>
245     /// <para>
246     /// The zero.
247     /// </para>
248     /// <para></para>
249     /// </summary>
250     public TLink this[TLink link, TLink index]
251     {
252         [MethodImpl(MethodImplOptions.AggressiveInlining)]
253         get
254         {
255             var root = GetTreeRoot(link);
256             if (GreaterOrEqualThan(index, GetSize(root)))
257             {
258                 return Zero;
259             }
260             while (!EqualToZero(root))
261             {
262                 var left = GetLeftOrDefault(root);
263                 var leftSize = GetSizeOrZero(left);
264                 if (LessThan(index, leftSize))
265                 {

```



```

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
    ↪ broken)
275 }
276 }
277
278 /// <summary>
279 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
280 /// </summary>
281 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
282 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
283 /// <returns>Индекс искомой связи.</returns>
284 [MethodImpl(MethodImplOptions.AggressiveInlining)]
285 public abstract TLink Search(TLink source, TLink target);
286
287 /// <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
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 }

```

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         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="InternalLinksSizeBalancedTreeMethodsBase"/> 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>

```

```

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,
86     ↪ 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 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) =>
    ↪ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));

220
221    /// <summary>
222    /// <para>
223    /// Gets the link values using the specified link index.

```

```

224     /// </para>
225     /// <para></para>
226     /// </summary>
227     /// <param name="linkIndex">
228     /// <para>The link index.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>A list of t link</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
237     {
238         ref var link = ref GetLinkDataPartReference(linkIndex);
239         return new Link<TLink>(linkIndex, link.Source, link.Target);
240     }
241
242     /// <summary>
243     /// <para>
244     /// The zero.
245     /// </para>
246     /// <para></para>
247     /// </summary>
248     public TLink this[TLink link, TLink index]
249     {
250         [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
279     /// <summary>
280     /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
281     /// ↪ (концом).
282     /// </summary>
283     /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
284     /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
285     /// <returns>Индекс искомой связи.</returns>
286     [MethodImpl(MethodImplOptions.AggressiveInlining)]
287     public abstract TLink Search(TLink source, TLink target);
288
289     /// <summary>
290     /// <para>
291     /// Searches the core using the specified root.
292     /// </para>
293     /// <para></para>
294     /// </summary>
295     /// <param name="root">
296     /// <para>The root.</para>
297     /// <para></para>
298     /// </param>
299     /// <param name="key">
300     /// <para>The key.</para>
301     /// <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,
365     ↪ GetTreeRoot(@base), handler);
366
367     // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
368     ↪ low-level MSIL stack.
369     [MethodImpl(MethodImplOptions.AggressiveInlining)]
370     private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink> handler)
371     {
372         var @continue = Continue;
373         if (EqualToZero(link))
374         {
375             return @continue;
376         }
377         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         /// </summary>
41         protected readonly TLink Continue;
42     }
43 }

```



```

35     /// </para>
36     /// <para></para>
37     /// </summary>
38     protected readonly TLink Continue;
39
40     /// <summary>
41     /// <para>
42     ///     Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     /// <param name="constants">
47     /// <para>A constants.</para>
48     /// <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     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     public InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants, byte*
60     ↪ linksDataParts, byte* linksIndexParts)
61     {
62         _linksDataParts = linksDataParts;
63         _linksIndexParts = linksIndexParts;
64         Break = constants.Break;
65         Continue = constants.Continue;
66     }
67
68     /// <summary>
69     /// <para>
70     ///     Gets the link data part reference using the specified link.
71     /// </para>
72     /// <para></para>
73     /// </summary>
74     /// <param name="link">
75     /// <para>The link.</para>
76     /// <para></para>
77     /// </param>
78     /// <returns>
79     /// <para>A ref raw link data part of t link</para>
80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
84     ↪ AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (RawLinkDataPart<TLink>.SizeInBytes
85     ↪ * _addressToInt64Converter.Convert(link)));
86
87     /// <summary>
88     /// <para>
89     ///     Gets the link index part reference using the specified link.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="link">
94     /// <para>The link.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>A ref raw link index part of t link</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
103    ↪ ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
104    ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
105
106    /// <summary>
107    /// <para>
108    ///     Gets the first using the specified head.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <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) =>
117        ↪ GetLinkIndexPartReference(head).RootAsSource;
118
119    /// <summary>
120    /// <para>
121    /// Gets the last using the specified head.
122    /// </para>
123    /// <para></para>
124    /// </summary>
125    /// <param name="head">
126    /// <para>The head.</para>
127    /// <para></para>
128    /// </param>
129    /// <returns>
130    /// <para>The link</para>
131    /// <para></para>
132    /// </returns>
133    [MethodImpl(MethodImplOptions.AggressiveInlining)]
134    protected override TLink GetLast(TLink head)
135    {
136        var first = GetLinkIndexPartReference(head).RootAsSource;
137        if (EqualToZero(first))
138        {
139            return first;
140        }
141        else
142        {
143            return GetPrevious(first);
144        }
145    }
146
147    /// <summary>
148    /// <para>
149    /// Gets the previous 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    /// <returns>
158    /// <para>The link</para>
159    /// <para></para>
160    /// </returns>
161    [MethodImpl(MethodImplOptions.AggressiveInlining)]
162    protected override TLink GetPrevious(TLink element) =>
163        ↪ GetLinkIndexPartReference(element).LeftAsSource;
164
165    /// <summary>
166    /// <para>
167    /// Gets the next using the specified element.
168    /// </para>
169    /// <para></para>
170    /// </summary>
171    /// <param name="element">
172    /// <para>The element.</para>
173    /// <para></para>
174    /// </param>
175    /// <returns>
176    /// <para>The link</para>
177    /// <para></para>
178    /// </returns>
179    [MethodImpl(MethodImplOptions.AggressiveInlining)]
180    protected override TLink GetNext(TLink element) =>
181        ↪ GetLinkIndexPartReference(element).RightAsSource;
182
183    /// <summary>
184    /// <para>
185    /// Gets the size using the specified head.

```

```

183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <param name="head">
187     /// <para>The head.</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 GetSize(TLink head) =>
196         ↪ GetLinkIndexPartReference(head).SizeAsSource;
197
198     /// <summary>
199     /// <para>
200     /// Sets the first using the specified head.
201     /// </para>
202     /// <para></para>
203     /// </summary>
204     /// <param name="head">
205     /// <para>The head.</para>
206     /// <para></para>
207     /// </param>
208     /// <param name="element">
209     /// <para>The element.</para>
210     /// <para></para>
211     /// </param>
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]
213     protected override void SetFirst(TLink head, TLink element) =>
214         ↪ GetLinkIndexPartReference(head).RootAsSource = element;
215
216     /// <summary>
217     /// <para>
218     /// Sets the last using the specified head.
219     /// </para>
220     /// <para></para>
221     /// </summary>
222     /// <param name="head">
223     /// <para>The head.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="element">
227     /// <para>The element.</para>
228     /// <para></para>
229     /// </param>
230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
231     protected override void SetLast(TLink head, TLink element)
232     {
233         //var first = GetLinkIndexPartReference(head).RootAsSource;
234         //if (EqualToZero(first))
235         //{
236             SetFirst(head, element);
237         //}
238         //else
239         //{
240             SetPrevious(first, element);
241         //}
242     }
243
244     /// <summary>
245     /// <para>
246     /// Sets the previous using the specified element.
247     /// </para>
248     /// <para></para>
249     /// </summary>
250     /// <param name="element">
251     /// <para>The element.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="previous">
255     /// <para>The previous.</para>
256     /// <para></para>
257     /// </param>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override void SetPrevious(TLink element, TLink previous) =>
260         ↪ GetLinkIndexPartReference(element).LeftAsSource = previous;

```

```

258
259     /// <summary>
260     /// <para>
261     /// Sets the next using the specified element.
262     /// </para>
263     /// <para></para>
264     /// </summary>
265     /// <param name="element">
266     /// <para>The element.</para>
267     /// <para></para>
268     /// </param>
269     /// <param name="next">
270     /// <para>The next.</para>
271     /// <para></para>
272     /// </param>
273     [MethodImpl(MethodImplOptions.AggressiveInlining)]
274     protected override void SetNext(TLink element, TLink next) =>
275         ↪ GetLinkIndexPartReference(element).RightAsSource = next;
276
277     /// <summary>
278     /// <para>
279     /// Sets the size using the specified head.
280     /// </para>
281     /// <para></para>
282     /// </summary>
283     /// <param name="head">
284     /// <para>The head.</para>
285     /// <para></para>
286     /// </param>
287     /// <param name="size">
288     /// <para>The size.</para>
289     /// <para></para>
290     /// </param>
291     [MethodImpl(MethodImplOptions.AggressiveInlining)]
292     protected override void SetSize(TLink head, TLink size) =>
293         ↪ GetLinkIndexPartReference(head).SizeAsSource = size;
294
295     /// <summary>
296     /// <para>
297     /// Counts the usages using the specified head.
298     /// </para>
299     /// <para></para>
300     /// </summary>
301     /// <param name="head">
302     /// <para>The head.</para>
303     /// <para></para>
304     /// </param>
305     /// <returns>
306     /// <para>The link</para>
307     /// <para></para>
308     /// </returns>
309     [MethodImpl(MethodImplOptions.AggressiveInlining)]
310     public TLink CountUsages(TLink head) => GetSize(head);
311
312     /// <summary>
313     /// <para>
314     /// Gets the link values using the specified link index.
315     /// </para>
316     /// <para></para>
317     /// </summary>
318     /// <param name="linkIndex">
319     /// <para>The link index.</para>
320     /// <para></para>
321     /// </param>
322     /// <returns>
323     /// <para>A list of t link</para>
324     /// <para></para>
325     /// </returns>
326     [MethodImpl(MethodImplOptions.AggressiveInlining)]
327     protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
328     {
329         ref var link = ref GetLinkDataPartReference(linkIndex);
330         return new Link<TLink>(linkIndex, link.Source, link.Target);
331     }
332
333     /// <summary>
334     /// <para>
335     /// 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    }
40 }

```

```

37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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 using the specified link.
183     /// </para>
184     /// <para></para>

```

```

182     /// </summary>
183     /// <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).RootAsSource;
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).Source;
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).Target;
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.LeftAsSource = Zero;
246         link.RightAsSource = Zero;
247         link.SizeAsSource = 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>

```



```

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) =>
        ↪ SearchCore(GetTreeRoot(source), target);
266 }
267 }

```

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> :
        ↪ 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,
        ↪ 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
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 }
270 }

```

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>

```

```

15 {
16     /// <summary>
17     /// <para>
18     /// Initializes a new <see
19     ↪ cref="InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
41     ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
42     ↪ base(constants, 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>
56     /// <para></para>
57     /// </returns>
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     protected override ref TLink GetLeftReference(TLink node) => ref
60     ↪ GetLinkIndexPartReference(node).LeftAsTarget;
61
62     /// <summary>
63     /// <para>
64     /// Gets the right reference using the specified node.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="node">
69     /// <para>The node.</para>
70     /// <para></para>
71     /// </param>
72     /// <returns>
73     /// <para>The ref link</para>
74     /// <para></para>
75     /// </returns>
76     [MethodImpl(MethodImplOptions.AggressiveInlining)]
77     protected override ref TLink GetRightReference(TLink node) => ref
78     ↪ GetLinkIndexPartReference(node).RightAsTarget;
79
80     /// <summary>
81     /// <para>
82     /// Gets the left using the specified node.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <param name="node">
87     /// <para>The node.</para>
88     /// <para></para>
89     /// </param>
90     /// <returns>
91     /// <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     /// </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).RightAsTarget;
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).LeftAsTarget = 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).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    /// </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).SizeAsTarget;

```

```

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>
169     /// <param name="size">
170     /// <para>The size.</para>
171     /// <para></para>
172     /// </param>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override void SetSize(TLink node, TLink size) =>
175     ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
176
177     /// <summary>
178     /// <para>
179     /// Gets the tree root using the specified link.
180     /// </para>
181     /// <para></para>
182     /// </summary>
183     /// <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>

```

```

233     /// </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.LeftAsTarget = Zero;
243         link.RightAsTarget = Zero;
244         link.SizeAsTarget = 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) =>
        ↪ 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)]

```



```

public InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
    ↪ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
    ↪ linksDataParts, linksIndexParts, header) { }

/// <summary>
/// <para>
/// Gets the left reference using the specified node.
/// </para>
/// <para></para>
/// </summary>
/// <param name="node">
/// <para>The node.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The ref link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override ref TLink GetLeftReference(TLink node) => ref
    ↪ GetLinkIndexPartReference(node).LeftAsTarget;

/// <summary>
/// <para>
/// Gets the right reference using the specified node.
/// </para>
/// <para></para>
/// </summary>
/// <param name="node">
/// <para>The node.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The ref link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override ref TLink GetRightReference(TLink node) => ref
    ↪ GetLinkIndexPartReference(node).RightAsTarget;

/// <summary>
/// <para>
/// Gets the left using the specified node.
/// </para>
/// <para></para>
/// </summary>
/// <param name="node">
/// <para>The node.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override TLink GetLeft(TLink node) =>
    ↪ GetLinkIndexPartReference(node).LeftAsTarget;

/// <summary>
/// <para>
/// Gets the right using the specified node.
/// </para>
/// <para></para>
/// </summary>
/// <param name="node">
/// <para>The node.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override TLink GetRight(TLink node) =>
    ↪ GetLinkIndexPartReference(node).RightAsTarget;

/// <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 }

```

1.45 ./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
            ↪ FileMappedResizableDirectMemory(dataMemory), new
            ↪ FileMappedResizableDirectMemory(indexMemory)) { }
44
45         /// <summary>
46         /// <para>
47         /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="dataMemory">
52         /// <para>A data memory.</para>
53         /// <para></para>
54         /// </param>
55         /// <param name="indexMemory">
56         /// <para>A index memory.</para>
57         /// <para></para>
58         /// </param>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
61
62         /// <summary>
63         /// <para>
64         /// Initializes a new <see cref="SplitMemoryLinks"/> instance.

```

```

65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="dataMemory">
69     /// <para>A data memory.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="indexMemory">
73     /// <para>A index memory.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="memoryReservationStep">
77     /// <para>A memory reservation step.</para>
78     /// <para></para>
79     /// </param>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↳ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
        ↳ IndexTreeType.Default, useLinkedList: true) { }

82     /// <summary>
83     /// <para>
84     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="dataMemory">
89     /// <para>A data memory.</para>
90     /// <para></para>
91     /// </param>
92     /// <param name="indexMemory">
93     /// <para>A index memory.</para>
94     /// <para></para>
95     /// </param>
96     /// <param name="memoryReservationStep">
97     /// <para>A memory reservation step.</para>
98     /// <para></para>
99     /// </param>
100    /// <param name="constants">
101    /// <para>A constants.</para>
102    /// <para></para>
103    /// </param>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    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 /// <para></para>
137 /// </param>
138 [MethodImpl(MethodImplOptions.AggressiveInlining)]
139 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
    ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
    ↪ memoryReservationStep, constants, useLinkedList)
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
            ↪ ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
152         _createInternalTargetTreeMethods = () => new
            ↪ InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
153         _createExternalTargetTreeMethods = () => new
            ↪ ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
154     }
155     Init(dataMemory, indexMemory);
156 }
157
158 /// <summary>
159 /// <para>
160 /// Sets the pointers using the specified data memory.
161 /// </para>
162 /// <para></para>
163 /// </summary>
164 /// <param name="dataMemory">
165 /// <para>The data memory.</para>
166 /// <para></para>
167 /// </param>
168 /// <param name="indexMemory">
169 /// <para>The index memory.</para>
170 /// <para></para>
171 /// </param>
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 protected override void SetPointers(IResizableDirectMemory dataMemory,
    ↪ IResizableDirectMemory indexMemory)
174 {
175     _linksDataParts = (byte*)dataMemory.Pointer;
176     _linksIndexParts = (byte*)indexMemory.Pointer;
177     _header = _linksIndexParts;
178     if (_useLinkedList)
179     {
180         InternalSourcesListMethods = new
            ↪ InternalLinksSourcesLinkedListMethods<TLink>(Constants, _linksDataParts,
            ↪ _linksIndexParts);
181     }
182     else
183     {
184         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
185     }
186     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
187     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
188     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
189     UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_linksDataParts, _header);
190 }

```

```

191
192     /// <summary>
193     /// <para>
194     /// Resets the pointers.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     protected override void ResetPointers()
200     {
201         base.ResetPointers();
202         _linksDataParts = null;
203         _linksIndexParts = null;
204         _header = null;
205     }
206
207     /// <summary>
208     /// <para>
209     /// Gets the header reference.
210     /// </para>
211     /// <para></para>
212     /// </summary>
213     /// <returns>
214     /// <para>A ref links header of t link</para>
215     /// <para></para>
216     /// </returns>
217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
218     protected override ref LinksHeader<TLink> GetHeaderReference() => ref
219         ↪ AsRef<LinksHeader<TLink>>(_header);
220
221     /// <summary>
222     /// <para>
223     /// Gets the link data part reference using the specified link index.
224     /// </para>
225     /// <para></para>
226     /// </summary>
227     /// <param name="linkIndex">
228     /// <para>The link index.</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         ↪ => ref AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (LinkDataPartSizeInBytes *
237         ↪ ConvertToInt64(linkIndex)));
238
239     /// <summary>
240     /// <para>
241     /// Gets the link index part reference using the specified link index.
242     /// </para>
243     /// <para></para>
244     /// </summary>
245     /// <param name="linkIndex">
246     /// <para>The link index.</para>
247     /// </param>
248     /// <returns>
249     /// <para>A ref raw link index part of t link</para>
250     /// <para></para>
251     /// </returns>
252     [MethodImpl(MethodImplOptions.AggressiveInlining)]
253     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
254         ↪ linkIndex) => ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
255         ↪ (LinkIndexPartSizeInBytes * ConvertToInt64(linkIndex)));
256 }

```

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>
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 LinkDataPartSizeInBytes = RawLinkDataPart<TLink>.SizeInBytes;
42
43         /// <summary>
44         /// <para>
45         /// The size in bytes.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         public static readonly long LinkIndexPartSizeInBytes =
50             RawLinkIndexPart<TLink>.SizeInBytes;
51
52         /// <summary>
53         /// <para>
54         /// The size in bytes.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
59
60         /// <summary>
61         /// <para>
62         /// The default links size step.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
67
68         /// <summary>
69         /// <para>
70         /// The data memory.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         protected readonly IResizableDirectMemory _dataMemory;
75
76         /// <summary>
77         /// <para>
78         /// The index memory.
79         /// </para>
80         /// <para></para>
81         /// </summary>
82         protected readonly IResizableDirectMemory _indexMemory;
83
84         /// <summary>
85         /// <para>
86         /// The use linked list.
87         /// </para>
88         /// <para></para>
89         /// </summary>

```



```

84     protected readonly bool _useLinkedList;
85     /// <summary>
86     /// <para>
87     /// The data memory reservation step in bytes.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     protected readonly long _dataMemoryReservationStepInBytes;
92     /// <summary>
93     /// <para>
94     /// The index memory reservation step in bytes.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     protected readonly long _indexMemoryReservationStepInBytes;
99
100    /// <summary>
101    /// <para>
102    /// The internal sources list methods.
103    /// </para>
104    /// <para></para>
105    /// </summary>
106    protected InternalLinksSourcesLinkedListMethods<TLink> InternalSourcesListMethods;
107    /// <summary>
108    /// <para>
109    /// The internal sources tree methods.
110    /// </para>
111    /// <para></para>
112    /// </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
146    /// <summary>
147    /// Возвращает общее число связей находящихся в хранилище.
148    /// </summary>
149    protected virtual TLink Total
150    {
151        [MethodImpl(MethodImplOptions.AggressiveInlining)]
152        get
153        {
154            {
155                ref var header = ref GetHeaderReference();
156                return Subtract(header.AllocatedLinks, header.FreeLinks);
157            }
158        }
159    }
160
161    /// <summary>
162    /// <para>
163    /// Gets the constants value.
164    /// </para>

```

```

161     /// <para></para>
162     </summary>
163     public virtual LinksConstants<TLink> Constants
164     {
165         [MethodImpl(MethodImplOptions.AggressiveInlining)]
166         get;
167     }
168
169     </summary>
170     <para>
171     </para>
172     </summary>
173     <para></para>
174     </summary>
175     <param name="dataMemory">
176     <para>A data memory.</para>
177     <para></para>
178     </param>
179     <param name="indexMemory">
180     <para>A index memory.</para>
181     <para></para>
182     </param>
183     <param name="memoryReservationStep">
184     <para>A memory reservation step.</para>
185     <para></para>
186     </param>
187     <param name="constants">
188     <para>A constants.</para>
189     <para></para>
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     </para>
209     </summary>
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     </para>
230     </summary>
231     <para></para>
232     </summary>

```

```

233 /// <param name="dataMemory">
234 /// <para>The data memory.</para>
235 /// </para>
236 /// </param>
237 /// <param name="indexMemory">
238 /// <para>The index memory.</para>
239 /// </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             {

```

```

375         return GetOne();
376     }
377     return GetZero();
378 }
379 }
380 if (restriction.Count == 3)
381 {
382     var externalReferencesRange = constants.ExternalReferencesRange;
383     var source = restriction[constants.SourcePart];
384     var target = restriction[constants.TargetPart];
385     if (AreEqual(index, any))
386     {
387         if (AreEqual(source, any) && AreEqual(target, any))
388         {
389             return Total;
390         }
391         else if (AreEqual(source, any))
392         {
393             if (externalReferencesRange.HasValue &&
394                 ⇨ externalReferencesRange.Value.Contains(target))
395             {
396                 return ExternalTargetsTreeMethods.CountUsages(target);
397             }
398             else
399             {
400                 return InternalTargetsTreeMethods.CountUsages(target);
401             }
402         }
403         else if (AreEqual(target, any))
404         {
405             if (externalReferencesRange.HasValue &&
406                 ⇨ externalReferencesRange.Value.Contains(source))
407             {
408                 return ExternalSourcesTreeMethods.CountUsages(source);
409             }
410             else
411             {
412                 if (_useLinkedList)
413                 {
414                     return InternalSourcesListMethods.CountUsages(source);
415                 }
416                 else
417                 {
418                     return InternalSourcesTreeMethods.CountUsages(source);
419                 }
420             }
421         }
422         else //if(source != Any && target != Any)
423         {
424             // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
425             TLink link;
426             if (externalReferencesRange.HasValue)
427             {
428                 if (externalReferencesRange.Value.Contains(source) &&
429                     ⇨ externalReferencesRange.Value.Contains(target))
430                 {
431                     link = ExternalSourcesTreeMethods.Search(source, target);
432                 }
433                 else if (externalReferencesRange.Value.Contains(source))
434                 {
435                     link = InternalTargetsTreeMethods.Search(source, target);
436                 }
437                 else if (externalReferencesRange.Value.Contains(target))
438                 {
439                     if (_useLinkedList)
440                     {
441                         link = ExternalSourcesTreeMethods.Search(source, target);
442                     }
443                     else
444                     {
445                         link = InternalSourcesTreeMethods.Search(source, target);
446                     }
447                 }
448             }
449             else
450             {
451                 if (_useLinkedList ||
452                     ⇨ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
453                     ⇨ 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) &&
            ↪ AreEqual(storedLinkValue.Target, target))
485         {
486             return GetOne();
487         }
488         return GetZero();
489     }
490     var value = default(TLink);
491     if (AreEqual(source, any))
492     {
493         value = target;
494     }
495     if (AreEqual(target, any))
496     {
497         value = source;
498     }
499     if (AreEqual(storedLinkValue.Source, value) ||
        ↪ AreEqual(storedLinkValue.Target, value))
500     {
501         return GetOne();
502     }
503     return GetZero();
504 }
505 }
506 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↪ поддерживаются.");
507 }
508
509 /// <summary>
510 /// <para>
511 /// Eaches the handler.
512 /// </para>
513 /// <para></para>
514 /// </summary>
515 /// <param name="handler">
516 /// <para>The handler.</para>
517 /// <para></para>
518 /// </param>
519 /// <param name="restriction">
520 /// <para>The substitution.</para>

```

```

521 /// <para></para>
522 /// </param>
523 /// <exception cref="NotSupportedException">
524 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
525 /// <para></para>
526 /// </exception>
527 /// <returns>
528 /// <para>The link</para>
529 /// <para></para>
530 /// </returns>
531 [MethodImpl(MethodImplOptions.AggressiveInlining)]
532 public virtual TLink Each(ICollection<TLink> restriction, ReadHandler<TLink> handler)
533 {
534     var constants = Constants;
535     var @break = constants.Break;
536     if (restriction.Count == 0)
537     {
538         for (var link = GetOne(); LessOrEqualThan(link,
539             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
540         {
541             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
542             {
543                 return @break;
544             }
545             return @break;
546         }
547     }
548     var @continue = constants.Continue;
549     var any = constants.Any;
550     var index = restriction[constants.IndexPart];
551     if (restriction.Count == 1)
552     {
553         if (AreEqual(index, any))
554         {
555             return Each(Array.Empty<TLink>(), handler);
556         }
557         if (!Exists(index))
558         {
559             return @continue;
560         }
561         return handler(GetLinkStruct(index));
562     }
563     if (restriction.Count == 2)
564     {
565         var value = restriction[1];
566         if (AreEqual(index, any))
567         {
568             if (AreEqual(value, any))
569             {
570                 return Each(Array.Empty<TLink>(), handler);
571             }
572             if (AreEqual(Each(new Link<TLink>(index, value, any), handler), @break))
573             {
574                 return @break;
575             }
576             return Each(new Link<TLink>(index, any, value), handler);
577         }
578         else
579         {
580             if (!Exists(index))
581             {
582                 return @continue;
583             }
584             if (AreEqual(value, any))
585             {
586                 return handler(GetLinkStruct(index));
587             }
588             ref var storedLinkValue = ref GetLinkDataPartReference(index);
589             if (AreEqual(storedLinkValue.Source, value) ||
590                 AreEqual(storedLinkValue.Target, value))
591             {
592                 return handler(GetLinkStruct(index));
593             }
594             return @continue;
595         }
596     }
597     if (restriction.Count == 3)
598     {

```

```

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
627         {
628             if (_useLinkedList)
629             {
630                 return InternalSourcesListMethods.EachUsage(source, handler);
631             }
632             else
633             {
634                 return InternalSourcesTreeMethods.EachUsage(source, handler);
635             }
636         }
637     }
638     else //if(source != Any && target != Any)
639     {
640         TLink link;
641         if (externalReferencesRange.HasValue)
642         {
643             if (externalReferencesRange.Value.Contains(source) &&
644                 ↪ externalReferencesRange.Value.Contains(target))
645             {
646                 link = ExternalSourcesTreeMethods.Search(source, target);
647             }
648             else if (externalReferencesRange.Value.Contains(source))
649             {
650                 link = InternalTargetsTreeMethods.Search(source, target);
651             }
652             else if (externalReferencesRange.Value.Contains(target))
653             {
654                 if (_useLinkedList)
655                 {
656                     link = ExternalSourcesTreeMethods.Search(source, target);
657                 }
658                 else
659                 {
660                     link = InternalSourcesTreeMethods.Search(source, target);
661                 }
662             }
663             else
664             {
665                 if (_useLinkedList ||
666                     ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
667                     ↪ InternalTargetsTreeMethods.CountUsages(target)))
668                 {
669                     link = InternalTargetsTreeMethods.Search(source, target);
670                 }
671                 else
672                 {
673                     link = InternalSourcesTreeMethods.Search(source, target);
674                 }
675             }
676         }
677     }
678 }

```



```

671     }
672     else
673     {
674         if (_useLinkedList ||
        ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
        ↪ InternalTargetsTreeMethods.CountUsages(target)))
        {
675             link = InternalTargetsTreeMethods.Search(source, target);
676         }
677         else
678         {
679             link = InternalSourcesTreeMethods.Search(source, target);
680         }
681     }
682     return AreEqual(link, constants.Null) ? @continue :
683     ↪ handler(GetLinkStruct(link));
684 }
685 }
686 else
687 {
688     if (!Exists(index))
689     {
690         return @continue;
691     }
692     if (AreEqual(source, any) && AreEqual(target, any))
693     {
694         return handler(GetLinkStruct(index));
695     }
696     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, IList<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
744 ↪ предварительно заполнено нулями
745 if (!AreEqual(source, @null))
746 {
747     if (externalReferencesRange.HasValue &&
748         ↪ externalReferencesRange.Value.Contains(source))
749     {
750         ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
751     }
752     else
753     {
754         if (_useLinkedList)
755         {
756             InternalSourcesListMethods.Detach(source, linkIndex);
757         }
758         else
759         {
760             InternalSourcesTreeMethods.Detach(ref
761                 ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
762         }
763     }
764 }
765 if (!AreEqual(target, @null))
766 {
767     if (externalReferencesRange.HasValue &&
768         ↪ externalReferencesRange.Value.Contains(target))
769     {
770         ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
771     }
772     else
773     {
774         InternalTargetsTreeMethods.Detach(ref
775             ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
776     }
777 }
778 source = link.Source = substitution[constants.SourcePart];
779 target = link.Target = substitution[constants.TargetPart];
780 if (!AreEqual(source, @null))
781 {
782     if (externalReferencesRange.HasValue &&
783         ↪ externalReferencesRange.Value.Contains(source))
784     {
785         ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
786     }
787     else
788     {
789         if (_useLinkedList)
790         {
791             InternalSourcesListMethods.AttachAsLast(source, linkIndex);
792         }
793         else
794         {
795             InternalSourcesTreeMethods.Attach(ref
796                 ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
797         }
798     }
799 }
800 if (!AreEqual(target, @null))
801 {
802     if (externalReferencesRange.HasValue &&
803         ↪ externalReferencesRange.Value.Contains(target))
804     {
805         ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
806     }
807     else
808     {
809         InternalTargetsTreeMethods.Attach(ref
810             ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
811     }
812 }
813 return handler != null ? handler(before, new Link<TLink>(linkIndex, source, target))
814     ↪ : Constants.Continue;
815 }
816
817 /// <remarks>
818 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
819 ↪ пространство
820 /// </remarks>

```

```

810 [MethodImpl(MethodImplOptions.AggressiveInlining)]
811 public virtual TLink Create(IList<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 /
833                 ↳ LinkDataPartSizeInBytes);
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(IList<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>

```

```

885 /// <param name="linkIndex">
886 /// <para>The link index.</para>
887 /// <para></para>
888 /// </param>
889 /// <returns>
890 /// <para>A list of t link</para>
891 /// <para></para>
892 /// </returns>
893 [MethodImpl(MethodImplOptions.AggressiveInlining)]
894 public IList<TLink> GetLinkStruct(TLink linkIndex)
895 {
896     ref var link = ref GetLinkDataPartReference(linkIndex);
897     return new Link<TLink>(linkIndex, link.Source, link.Target);
898 }
899
900 /// <remarks>
901 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
902   → адрес реально поменялся
903 ///
904 /// Указатель this.links может быть в том же месте,
905 /// так как 0-я связь не используется и имеет такой же размер как Header,
906 /// поэтому header размещается в том же месте, что и 0-я связь
907 /// </remarks>
908 [MethodImpl(MethodImplOptions.AggressiveInlining)]
909 protected abstract void SetPointers(IResizableDirectMemory dataMemory,
910   → IResizableDirectMemory indexMemory);
911
912 /// <summary>
913 /// <para>
914 /// Resets the pointers.
915 /// </para>
916 /// <para></para>
917 /// </summary>
918 [MethodImpl(MethodImplOptions.AggressiveInlining)]
919 protected virtual void ResetPointers()
920 {
921     InternalSourcesListMethods = null;
922     InternalSourcesTreeMethods = null;
923     ExternalSourcesTreeMethods = null;
924     InternalTargetsTreeMethods = null;
925     ExternalTargetsTreeMethods = null;
926     UnusedLinksListMethods = null;
927 }
928
929 /// <summary>
930 /// <para>
931 /// Gets the header reference.
932 /// </para>
933 /// <para></para>
934 /// </summary>
935 /// <returns>
936 /// <para>A ref links header of t link</para>
937 /// <para></para>
938 /// </returns>
939 [MethodImpl(MethodImplOptions.AggressiveInlining)]
940 protected abstract ref LinksHeader<TLink> GetHeaderReference();
941
942 /// <summary>
943 /// <para>
944 /// Gets the link data part reference using the specified link index.
945 /// </para>
946 /// <para></para>
947 /// </summary>
948 /// <param name="linkIndex">
949 /// <para>The link index.</para>
950 /// <para></para>
951 /// </param>
952 /// <returns>
953 /// <para>A ref raw link data part of t link</para>
954 /// <para></para>
955 /// </returns>
956 [MethodImpl(MethodImplOptions.AggressiveInlining)]
957 protected abstract ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex);
958
959 /// <summary>
960 /// <para>
961 /// Gets the link index part reference using the specified link index.
962 /// </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     /// <summary>
974     /// <para>
975     /// Determines whether this instance exists.
976     /// </para>
977     /// <para></para>
978     /// </summary>
979     /// <param name="link">
980     /// <para>The link.</para>
981     /// <para></para>
982     /// </param>
983     /// <returns>
984     /// <para>The bool</para>
985     /// <para></para>
986     /// </returns>
987     [MethodImpl(MethodImplOptions.AggressiveInlining)]
988     protected virtual bool Exists(TLink link)
989         => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
990         && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
991         && !IsUnusedLink(link);
992
993     /// <summary>
994     /// <para>
995     /// Determines whether this instance is unused link.
996     /// </para>
997     /// <para></para>
998     /// </summary>
999     /// <param name="linkIndex">
1000     /// <para>The link index.</para>
1001     /// <para></para>
1002     /// </param>
1003     /// <returns>
1004     /// <para>The bool</para>
1005     /// <para></para>
1006     /// </returns>
1007     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1008     protected virtual bool IsUnusedLink(TLink linkIndex)
1009     {
1010         if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
1011             ↪ is not needed
1012         {
1013             // TODO: Reduce access to memory in different location (should be enough to use
1014             ↪ just linkIndexPart)
1015             ref var linkDataPart = ref GetLinkDataPartReference(linkIndex);
1016             ref var linkIndexPart = ref GetLinkIndexPartReference(linkIndex);
1017             return AreEqual(linkIndexPart.SizeAsTarget, default) &&
1018                 ↪ !AreEqual(linkDataPart.Source, default);
1019         }
1020         else
1021         {
1022             return true;
1023         }
1024     }
1025
1026     /// <summary>
1027     /// <para>
1028     /// Gets the one.
1029     /// </para>
1030     /// <para></para>
1031     /// </summary>
1032     /// <returns>
1033     /// <para>The link</para>
1034     /// <para></para>
1035     /// </returns>
1036     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

1035     protected virtual TLink GetOne() => _one;
1036
1037     /// <summary>
1038     /// <para>
1039     /// Gets the zero.
1040     /// </para>
1041     /// <para></para>
1042     /// </summary>
1043     /// <returns>
1044     /// <para>The link</para>
1045     /// <para></para>
1046     /// </returns>
1047     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1048     protected virtual TLink GetZero() => default;
1049
1050     /// <summary>
1051     /// <para>
1052     /// Determines whether this instance are equal.
1053     /// </para>
1054     /// <para></para>
1055     /// </summary>
1056     /// <param name="first">
1057     /// <para>The first.</para>
1058     /// <para></para>
1059     /// </param>
1060     /// <param name="second">
1061     /// <para>The second.</para>
1062     /// <para></para>
1063     /// </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)]

```

```

1111     protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
1112         ↪ _comparer.Compare(first, second) <= 0;
1113
1114     /// <summary>
1115     /// <para>
1116     /// Determines whether this instance greater than.
1117     /// </para>
1118     /// </summary>
1119     /// <param name="first">
1120     /// <para>The first.</para>
1121     /// <para></para>
1122     /// </param>
1123     /// <param name="second">
1124     /// <para>The second.</para>
1125     /// <para></para>
1126     /// </param>
1127     /// <returns>
1128     /// <para>The bool</para>
1129     /// <para></para>
1130     /// </returns>
1131     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1132     protected virtual bool GreaterThan(TLink first, TLink second) =>
1133         ↪ _comparer.Compare(first, second) > 0;
1134
1135     /// <summary>
1136     /// <para>
1137     /// Determines whether this instance greater or equal than.
1138     /// </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     /// </summary>
1161     /// <param name="value">
1162     /// <para>The value.</para>
1163     /// <para></para>
1164     /// </param>
1165     /// <returns>
1166     /// <para>The long</para>
1167     /// <para></para>
1168     /// </returns>
1169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1170     protected virtual long ConvertToInt64(TLink value) =>
1171         ↪ _addressToInt64Converter.Convert(value);
1172
1173     /// <summary>
1174     /// <para>
1175     /// Converts the to address using the specified value.
1176     /// </para>
1177     /// </summary>
1178     /// <param name="value">
1179     /// <para>The value.</para>
1180     /// <para></para>
1181     /// </param>
1182     /// <returns>
1183     /// <para>The link</para>
1184     /// <para></para>

```

```

1185     /// </returns>
1186     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1187     protected virtual TLink ConvertToAddress(long value) =>
1188         ↪ _int64ToAddressConverter.Convert(value);
1189
1189     /// <summary>
1190     /// <para>
1191     /// Adds the first.
1192     /// </para>
1193     /// <para></para>
1194     /// </summary>
1195     /// <param name="first">
1196     /// <para>The first.</para>
1197     /// <para></para>
1198     /// </param>
1199     /// <param name="second">
1200     /// <para>The second.</para>
1201     /// <para></para>
1202     /// </param>
1203     /// <returns>
1204     /// <para>The link</para>
1205     /// <para></para>
1206     /// </returns>
1207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1208     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
1209         ↪ second);
1210
1210     /// <summary>
1211     /// <para>
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) =>
1230         ↪ Arithmetic<TLink>.Subtract(first, second);
1231
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>

```

```

29     /// </summary>
30     /// <param name="links">
31     /// <para>A links.</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 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     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
57     ↪ AsRef<LinksHeader<TLink>>(_header);
58
59     /// <summary>
60     /// <para>
61     /// Gets the link data part reference using the specified link.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="link">
66     /// <para>The link.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>A ref raw link data part of t link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
75     ↪ AsRef<RawLinkDataPart<TLink>>(_links + (RawLinkDataPart<TLink>.SizeInBytes *
76     ↪ _addressToInt64Converter.Convert(link)));
77
78     /// <summary>
79     /// <para>
80     /// Gets the first.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <returns>
85     /// <para>The link</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
90
91     /// <summary>
92     /// <para>
93     /// Gets the last.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <returns>
98     /// <para>The link</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
103
104    /// <summary>
105    /// <para>
106    /// 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) =>
117        ↪ GetLinkDataPartReference(element).Source;
118
119    /// <summary>
120    /// <para>
121    /// Gets the next using the specified element.
122    /// </para>
123    /// <para></para>
124    /// </summary>
125    /// <param name="element">
126    /// <para>The element.</para>
127    /// <para></para>
128    /// </param>
129    /// <returns>
130    /// <para>The link</para>
131    /// <para></para>
132    /// </returns>
133    [MethodImpl(MethodImplOptions.AggressiveInlining)]
134    protected override TLink GetNext(TLink element) =>
135        ↪ GetLinkDataPartReference(element).Target;
136
137    /// <summary>
138    /// <para>
139    /// Gets the size.
140    /// </para>
141    /// <para></para>
142    /// </summary>
143    /// <returns>
144    /// <para>The link</para>
145    /// <para></para>
146    /// </returns>
147    [MethodImpl(MethodImplOptions.AggressiveInlining)]
148    protected override TLink GetSize() => GetHeaderReference().FreeLinks;
149
150    /// <summary>
151    /// <para>
152    /// Sets the first using the specified element.
153    /// </para>
154    /// <para></para>
155    /// </summary>
156    /// <param name="element">
157    /// <para>The element.</para>
158    /// <para></para>
159    /// </param>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
161    protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
162        ↪ element;
163
164    /// <summary>
165    /// <para>
166    /// Sets the last 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    [MethodImpl(MethodImplOptions.AggressiveInlining)]
175    protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
176        ↪ element;
177
178    /// <summary>
179    /// <para>
180    /// Sets the previous using the specified element.
181    /// </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) =>
190         ↪ GetLinkDataPartReference(element).Source = previous;
191
192     /// <summary>
193     /// <para>
194     /// Sets the next using the specified element.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="element">
199     /// <para>The element.</para>
200     /// <para></para>
201     /// </param>
202     /// <param name="next">
203     /// <para>The next.</para>
204     /// <para></para>
205     /// </param>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override void SetNext(TLink element, TLink next) =>
208         ↪ GetLinkDataPartReference(element).Target = next;
209
210     /// <summary>
211     /// <para>
212     /// Sets the size using the specified size.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     /// <param name="size">
217     /// <para>The size.</para>
218     /// <para></para>
219     /// </param>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
222 }

```

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.

```

```

31     /// </para>
32     /// <para></para>
33     /// </summary>
34     public TLink Source;
35     /// <summary>
36     /// <para>
37     /// The target.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public TLink Target;
42
43     /// <summary>
44     /// <para>
45     /// Determines whether this instance equals.
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 bool</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public override bool Equals(object obj) => obj is RawLinkDataPart<TLink> link ?
59         ↪ Equals(link) : false;
60
61     /// <summary>
62     /// <para>
63     /// Determines whether this instance equals.
64     /// </para>
65     /// <para></para>
66     /// </summary>
67     /// <param name="other">
68     /// <para>The other.</para>
69     /// <para></para>
70     /// </param>
71     /// <returns>
72     /// <para>The bool</para>
73     /// <para></para>
74     /// </returns>
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     public bool Equals(RawLinkDataPart<TLink> other)
77         => _equalityComparer.Equals(Source, other.Source)
78         && _equalityComparer.Equals(Target, other.Target);
79
80     /// <summary>
81     /// <para>
82     /// Gets the hash code.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <returns>
87     /// <para>The int</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public override int GetHashCode() => (Source, Target).GetHashCode();
92
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     public static bool operator ==(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
95         ↪ right) => left.Equals(right);
96
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     public static bool operator !=(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
99         ↪ right) => !(left == right);
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;
36
37         /// <summary>
38         /// <para>
39         /// The left as source.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink LeftAsSource;
44
45         /// <summary>
46         /// <para>
47         /// The right as source.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public TLink RightAsSource;
52
53         /// <summary>
54         /// <para>
55         /// The size as source.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         public TLink SizeAsSource;
60
61         /// <summary>
62         /// <para>
63         /// The root as target.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         public TLink RootAsTarget;
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         /// <summary>
86         /// <para>
87         /// The size as target.
88         /// </para>
89         /// <para></para>
90         /// </summary>
91         public TLink SizeAsTarget;
92     }
93 }

```

```

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,
    ↪ SizeAsSource, RootAsTarget, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
139
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    public static bool operator ==(RawLinkIndexPart<TLink> left, RawLinkIndexPart<TLink>
    ↪ right) => left.Equals(right);
142
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    public static bool operator !=(RawLinkIndexPart<TLink> left, RawLinkIndexPart<TLink>
    ↪ right) => !(left == right);
145 }
146 }

```

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 :
    ↳ 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="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)]

```



```

83     protected override TLink GetZero() => 0U;
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(TLink value) => value == 0U;
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(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>

```

```

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) =>
397         ↪ ref LinksIndexParts[link];
398
399     /// <summary>
400     /// <para>
401     /// Determines whether this instance first is to the left of second.
402     /// </para>
403     /// </para></para>
404     /// </summary>
405     /// <param name="first">
406     /// <para>The first.</para>
407     /// </para></param>
408     /// <param name="second">
409     /// <para>The second.</para>
410     /// </para></param>
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,
422             ↪ secondLink.Source, secondLink.Target);
423     }
424
425     /// <summary>
426     /// <para>
427     /// Determines whether this instance first is to the right of second.
428     /// </para>
429     /// </para></para>
430     /// </summary>
431     /// <param name="first">
432     /// <para>The first.</para>
433     /// </para></param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// </para></param>
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 }

```

1.51 ./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"/>
    ↳ 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 UInt32ExternalLinksSizeBalancedTreeMethodsBase(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 link</para>
79      /// <para></para>
80      /// </returns>
81      [MethodImpl(MethodImplOptions.AggressiveInlining)]
82      protected override TLink GetZero() => 0U;
83
84      /// <summary>
85      /// <para>
86      /// Determines whether this instance equal to zero.
87

```

```

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(TLink value) => value == 0U;
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(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)]

```

```

396     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
397         ↪ ref LinksIndexParts[link];
398
399     /// <summary>
400     /// <para>
401     /// Determines whether this instance first is to the left of second.
402     /// </para>
403     /// </summary>
404     /// <param name="first">
405     /// <para>The first.</para>
406     /// </param>
407     /// <param name="second">
408     /// <para>The second.</para>
409     /// </param>
410     /// <returns>
411     /// <para>The bool</para>
412     /// </returns>
413     [MethodImpl(MethodImplOptions.AggressiveInlining)]
414     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
415     {
416         ref var firstLink = ref LinksDataParts[first];
417         ref var secondLink = ref LinksDataParts[second];
418         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
419             ↪ secondLink.Source, secondLink.Target);
420     }
421
422     /// <summary>
423     /// <para>
424     /// Determines whether this instance first is to the right of second.
425     /// </para>
426     /// </summary>
427     /// <param name="first">
428     /// <para>The first.</para>
429     /// </param>
430     /// <param name="second">
431     /// <para>The second.</para>
432     /// </param>
433     /// <returns>
434     /// <para>The bool</para>
435     /// </returns>
436     [MethodImpl(MethodImplOptions.AggressiveInlining)]
437     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
438     {
439         ref var firstLink = ref LinksDataParts[first];
440         ref var secondLink = ref LinksDataParts[second];
441         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
442             ↪ secondLink.Source, secondLink.Target);
443     }
444 }
445
446 }
447
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     /// </summary>
13     /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
15         ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>

```

```

18    /// <para>
19    /// Initializes a new <see
    ↪ cref="UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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
    ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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>

```

```

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     /// </summary>
184     /// <returns>
185     /// <para>The link</para>
186     /// <para></para>
187     /// </returns>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override TLink GetTreeRoot() => Header->RootAsSource;
190
191     /// <summary>
192     /// <para>
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 FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
236         ↪ TLink secondSource, TLink secondTarget)
237         => firstSource < secondSource || firstSource == secondSource && firstTarget <
238         ↪ secondTarget;
239
240     /// <summary>
241     /// <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     => firstSource > secondSource || firstSource == secondSource && firstTarget >
268     ↪ secondTarget;
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.LeftAsSource = Zero;
285         link.RightAsSource = Zero;
286         link.SizeAsSource = Zero;
287     }
288 }

```

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>

```

```

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 UInt32 ExternalLinksSourcesSizeBalancedTreeMethods(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>

```

```

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     /// </summary>
184     /// <returns>
185     /// <para>The link</para>
186     /// <para></para>
187     /// </returns>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override TLink GetTreeRoot() => Header->RootAsSource;
190
191     /// <summary>
192     /// <para>
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 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">

```

```

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         => firstSource > secondSource || firstSource == secondSource && firstTarget >
268         ↪ secondTarget;
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.LeftAsSource = Zero;
285         link.RightAsSource = Zero;
286         link.SizeAsSource = Zero;
287     }
288 }
289 }

```

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>

```

```

34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public
41    ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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    /// <para></para>
51    /// </summary>
52    /// <param name="node">
53    /// <para>The node.</para>
54    /// <para></para>
55    /// </param>
56    /// <returns>
57    /// <para>The ref link</para>
58    /// <para></para>
59    /// </returns>
60    [MethodImpl(MethodImplOptions.AggressiveInlining)]
61    protected override ref TLink GetLeftReference(TLink node) => ref
62    ↪ LinksIndexParts[node].LeftAsTarget;
63
64    /// <summary>
65    /// <para>
66    /// Gets the right reference using the specified node.
67    /// </para>
68    /// <para></para>
69    /// </summary>
70    /// <param name="node">
71    /// <para>The node.</para>
72    /// <para></para>
73    /// </param>
74    /// <returns>
75    /// <para>The ref link</para>
76    /// <para></para>
77    /// </returns>
78    [MethodImpl(MethodImplOptions.AggressiveInlining)]
79    protected override ref TLink GetRightReference(TLink node) => ref
80    ↪ LinksIndexParts[node].RightAsTarget;
81
82    /// <summary>
83    /// <para>
84    /// Gets the left using the specified node.
85    /// </para>
86    /// <para></para>
87    /// </summary>
88    /// <param name="node">
89    /// <para>The node.</para>
90    /// <para></para>
91    /// </param>
92    /// <returns>
93    /// <para>The link</para>
94    /// <para></para>
95    /// </returns>
96    [MethodImpl(MethodImplOptions.AggressiveInlining)]
97    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
98
99    /// <summary>
100    /// <para>
101    /// Gets the right using the specified node.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    /// <param name="node">
106    /// <para>The node.</para>
107    /// <para></para>
108    /// </param>
109    /// <returns>
110    /// <para>The link</para>
111    /// <para></para>
112    /// </returns>

```

```

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)]
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.

```

```

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() => Header->RootAsTarget;
190
191     /// <summary>
192     /// <para>
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].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)
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">

```

```

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 }

```

1.55 ./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)]

```

```

40 public UInt32ExternalLinksTargetsSizeBalancedTreeMethods(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)]
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.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <returns>
188     /// <para>The link</para>
189     /// <para></para>
190     /// </returns>

```

```

188 [MethodImpl(MethodImplOptions.AggressiveInlining)]
189 protected override TLink GetTreeRoot() => Header->RootAsTarget;
190
191 /// <summary>
192 /// <para>
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].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)
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>

```

```

264 [MethodImpl(MethodImplOptions.AggressiveInlining)]
265 protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
    ↳ TLink secondSource, TLink secondTarget)
266 => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
    ↳ secondSource;
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.LeftAsTarget = Zero;
283     link.RightAsTarget = Zero;
284     link.SizeAsTarget = Zero;
285 }
286 }
287 }

```

1.56 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase

```

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 :
    ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
17    {
18        /// <summary>
19        /// <para>
20        /// The links data parts.
21        /// </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
    ↳ cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase" /> instance.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="constants">
47        /// <para>A constants.</para>
48        /// <para></para>
49        /// </param>

```

```

50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// </para>
53     /// </param>
54     /// <param name="linksIndexParts">
55     /// <para>A links index parts.</para>
56     /// </para>
57     /// </param>
58     /// <param name="header">
59     /// <para>A header.</para>
60     /// </para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected
64     ↪ UInt32 InternalLinksRecursionlessSizeBalancedTreeMethodsBase(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     /// </summary>
79     /// <returns>
80     /// <para>The link</para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override TLink GetZero() => 0U;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equal to zero.
88     /// </para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// </para>
93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// </returns>
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     protected override bool EqualToZero(TLink value) => value == 0U;
99
100     /// <summary>
101     /// <para>
102     /// Determines whether this instance are equal.
103     /// </para>
104     /// </summary>
105     /// <param name="first">
106     /// <para>The first.</para>
107     /// </para>
108     /// </param>
109     /// <param name="second">
110     /// <para>The second.</para>
111     /// </para>
112     /// </param>
113     /// <returns>
114     /// <para>The bool</para>
115     /// </returns>
116     [MethodImpl(MethodImplOptions.AggressiveInlining)]
117     protected override bool AreEqual(TLink first, TLink second) => first == second;
118
119     /// <summary>
120     /// <para>
121     /// 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
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(TLink 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(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>
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)]
405     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
406         ↪ GetKeyPartValue(first) < GetKeyPartValue(second);
407
408     /// <summary>
409     /// <para>
410     /// Determines whether this instance first is to the right of second.
411     /// </para>
412     /// <para></para>
413     /// </summary>
414     /// <param name="first">
415     /// <para>The first.</para>
416     /// <para></para>
417     /// </param>
418     /// <param name="second">
419     /// <para>The second.</para>
420     /// <para></para>
421     /// </param>
422     /// <returns>
423     /// <para>The bool</para>
424     /// <para></para>
425     /// </returns>
426     [MethodImpl(MethodImplOptions.AggressiveInlining)]
427     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
428         ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
429 }

```


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         /// <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="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
44         ↪ 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 UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
66         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
67         ↪ linksIndexParts, LinksHeader<TLink>* header)
68         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
69         {
70             LinksDataParts = linksDataParts;
71             LinksIndexParts = linksIndexParts;
72             Header = header;
73         }
74
75         /// <summary>
76         /// <para>
77         /// Gets the zero.
78         /// </para>

```

```

75     /// <para></para>
76     /// </summary>
77     /// <returns>
78     /// <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
197     ↪ 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
215     ↪ always >= 0 for ulong
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>

```

```

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)]

```

```

382     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
383         ↪ ref LinksIndexParts[link];
384
385     /// <summary>
386     /// <para>
387     /// Determines whether this instance first is to the left of second.
388     /// </para>
389     /// </summary>
390     /// <param name="first">
391     /// <para>The first.</para>
392     /// </param>
393     /// <param name="second">
394     /// <para>The second.</para>
395     /// </param>
396     /// <returns>
397     /// <para>The bool</para>
398     /// </returns>
399     [MethodImpl(MethodImplOptions.AggressiveInlining)]
400     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
401         ↪ GetKeyPartValue(first) < GetKeyPartValue(second);
402
403     /// <summary>
404     /// <para>
405     /// Determines whether this instance first is to the right of second.
406     /// </para>
407     /// </summary>
408     /// <param name="first">
409     /// <para>The first.</para>
410     /// </param>
411     /// <param name="second">
412     /// <para>The second.</para>
413     /// </param>
414     /// <returns>
415     /// <para>The bool</para>
416     /// </returns>
417     [MethodImpl(MethodImplOptions.AggressiveInlining)]
418     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
419         ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
420 }
421 }
422

```

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     /// </summary>
13     /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLink}"/>
14     public unsafe class UInt32InternalLinksSourcesLinkedListMethods :
15         ↪ 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         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27

```

```

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,
40     ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
41     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
42     {
43         _linksDataParts = linksDataParts;
44         _linksIndexParts = linksIndexParts;
45     }
46     /// <summary>
47     /// <para>
48     /// Gets the link data part reference using the specified link.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="link">
53     /// <para>The link.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>A ref raw link data part of t link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
62     ↪ ref _linksDataParts[link];
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) =>
79     ↪ ref _linksIndexParts[link];
80 }

```

1.59 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods

```

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 :
16     ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <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>

```

```

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.
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];

```

```

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) =>
        ↪ SearchCore(GetTreeRoot(source), target);
267 }
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"/>
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="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 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    /// </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    /// </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 using the specified node.
182    /// </para>
183    /// <para></para>
184    /// </summary>
185    /// <param name="node">
186    /// <para>The node.</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].RootAsSource;
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].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 }

```

1.61 ./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>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public
43        ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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)]

```

```

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

```

```

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"/>
    ↪ 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 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
    ↪ 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)]
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>

```

```

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) =>
        ↪ SearchCore(GetTreeRoot(target), source);
267 }
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         /// <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 UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46         /// <summary>
47         /// <para>
48         /// Initializes a new <see cref="UInt32SplitMemoryLinks" /> 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 UInt32SplitMemoryLinks(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="UInt32SplitMemoryLinks"/> 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 UInt32SplitMemoryLinks(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="UInt32SplitMemoryLinks"/> 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 UInt32SplitMemoryLinks(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
            ↳ UInt32InternalLinksSourcesSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
128         _createExternalSourceTreeMethods = () => new
            ↳ UInt32ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
    }

```

```

129         _createInternalTargetTreeMethods = () => new
130         ↪ UInt32InternalLinksTargetsSizeBalancedTreeMethods(Constants,
131         ↪ _linksDataParts, _linksIndexParts, _header);
132     _createExternalTargetTreeMethods = () => new
133     ↪ UInt32ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
134     ↪ _linksDataParts, _linksIndexParts, _header);
135 }
136 else
137 {
138     _createInternalSourceTreeMethods = () => new
139     ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
140     ↪ _linksDataParts, _linksIndexParts, _header);
141     _createExternalSourceTreeMethods = () => new
142     ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
143     ↪ _linksDataParts, _linksIndexParts, _header);
144     _createInternalTargetTreeMethods = () => new
145     ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
146     ↪ _linksDataParts, _linksIndexParts, _header);
147     _createExternalTargetTreeMethods = () => new
148     ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
149     ↪ _linksDataParts, _linksIndexParts, _header);
150 }
151 Init(dataMemory, indexMemory);
152 }
153
154 /// <summary>
155 /// <para>
156 /// Sets the pointers using the specified data memory.
157 /// </para>
158 /// <para></para>
159 /// </summary>
160 /// <param name="dataMemory">
161 /// <para>The data memory.</para>
162 /// <para></para>
163 /// </param>
164 /// <param name="indexMemory">
165 /// <para>The index memory.</para>
166 /// <para></para>
167 /// </param>
168 [MethodImpl(MethodImplOptions.AggressiveInlining)]
169 protected override void SetPointers(IResizableDirectMemory dataMemory,
170 ↪ IResizableDirectMemory indexMemory)
171 {
172     _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
173     _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
174     _header = (LinksHeader<TLink>*)indexMemory.Pointer;
175     if (_useLinkedList)
176     {
177         InternalSourcesListMethods = new
178         ↪ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
179         ↪ _linksIndexParts);
180     }
181     else
182     {
183         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
184     }
185     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
186     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
187     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
188     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
189 }
190
191 /// <summary>
192 /// <para>
193 /// Resets the pointers.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 [MethodImpl(MethodImplOptions.AggressiveInlining)]
198 protected override void ResetPointers()
199 {
200     base.ResetPointers();
201     _linksDataParts = null;
202     _linksIndexParts = null;
203     _header = null;
204 }
205
206 /// <summary>

```



```

192    /// <para>
193    /// Gets the header reference.
194    /// </para>
195    /// <para></para>
196    /// </summary>
197    /// <returns>
198    /// <para>A ref links header of t link</para>
199    /// <para></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    /// <para></para>
209    /// </summary>
210    /// <param name="linkIndex">
211    /// <para>The link index.</para>
212    /// <para></para>
213    /// </param>
214    /// <returns>
215    /// <para>A ref raw link data part of t link</para>
216    /// <para></para>
217    /// </returns>
218    [MethodImpl(MethodImplOptions.AggressiveInlining)]
219    protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
220    ↪ => ref _linksDataParts[linkIndex];
221
222    /// <summary>
223    /// <para>
224    /// Gets the link index part reference 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 ref raw link index part of t link</para>
234    /// <para></para>
235    /// </returns>
236    [MethodImpl(MethodImplOptions.AggressiveInlining)]
237    protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
238    ↪ linkIndex) => ref _linksIndexParts[linkIndex];
239
240    /// <summary>
241    /// <para>
242    /// Determines whether this instance are equal.
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 AreEqual(TLink first, TLink second) => first == second;
260
261    /// <summary>
262    /// <para>
263    /// Determines whether this instance less than.
264    /// </para>
265    /// <para></para>
266    /// </summary>
267    /// <param name="first">
268    /// <para>The first.</para>
269    /// <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(TLink first, TLink 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(TLink first, TLink 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(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     /// </summary>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// </param>
29     /// <param name="header">
30     /// <para>A header.</para>
31     /// </param>
32     [MethodImpl(MethodImplOptions.AggressiveInlining)]
33     public UInt32UnusedLinksListMethods(RawLinkDataPart<TLink>* links, LinksHeader<TLink>*
34     ↪ header)
35     : base((byte*)links, (byte*)header)
36     {
37         _links = links;
38         _header = header;
39     }
40
41     /// <summary>
42     /// <para>
43     /// Gets the link data part reference using the specified link.
44     /// </para>
45     /// </summary>
46     /// <param name="link">
47     /// <para>The link.</para>
48     /// </param>
49     /// <returns>
50     /// <para>A ref raw link data part of t link</para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
54     ↪ ref _links[link];
55
56     /// <summary>
57     /// <para>
58     /// Gets the header reference.
59     /// </para>
60     /// </summary>
61     /// <returns>
62     /// <para>A ref links header of t link</para>
63     /// </returns>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
66 }
67
68 }
69
70 }
71
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     /// </summary>
14     /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
15     /// <seealso cref="ILinksTreeMethods{TLink}"/>
16     public unsafe abstract class UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17     ↪ 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     /// <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     ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(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 ulong</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected override ulong GetZero() => 0UL;
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>

```

```

96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100     protected override bool EqualToZero(ulong value) => value == 0UL;
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 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>

```

```

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,
422             ↪ secondLink.Source, secondLink.Target);
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,
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;

```

```

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="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
44    ↪ 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 UInt64ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
66    ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
67    ↪ linksIndexParts, LinksHeader<TLink>* header)
68    : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
69    {
70    LinksDataParts = linksDataParts;
71    LinksIndexParts = linksIndexParts;
72    Header = header;
73    }
74
75    /// <summary>
76    /// <para>
77    /// Gets the zero.
78    /// </para>
79    /// <para></para>
80    /// </summary>
81    /// <returns>
82    /// <para>The ulong</para>
83    /// <para></para>
84    /// </returns>
85    [MethodImpl(MethodImplOptions.AggressiveInlining)]
86    protected override ulong GetZero() => 0UL;
87
88    /// <summary>
89    /// <para>
90    /// Determines whether this instance equal to zero.
91    /// </para>
92    /// <para></para>
93    /// </summary>
94    /// <param name="value">
95    /// <para>The value.</para>
96    /// <para></para>
97    /// </param>
98    /// <returns>
99    /// <para>The bool</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override bool EqualToZero(ulong value) => value == 0UL;

```

```

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 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>

```



```

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,
422             ↪ secondLink.Source, secondLink.Target);
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,
448             ↪ secondLink.Source, secondLink.Target);
449     }
450 }

```

1.67 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethodsBase

```

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>

```

```

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
    ↪ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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;

```

```

177     /// <summary>
178     /// <para>
179     /// Gets the tree root.
180     /// </para>
181     /// <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() => Header->RootAsSource;
189
190     /// <summary>
191     /// <para>
192     /// Gets the base part value 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     /// <returns>
201     /// <para>The link</para>
202     /// <para></para>
203     /// </returns>
204     [MethodImpl(MethodImplOptions.AggressiveInlining)]
205     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
206
207     /// <summary>
208     /// <para>
209     /// Determines whether this instance first is to the left of second.
210     /// </para>
211     /// <para></para>
212     /// </summary>
213     /// <param name="firstSource">
214     /// <para>The first source.</para>
215     /// <para></para>
216     /// </param>
217     /// <param name="firstTarget">
218     /// <para>The first target.</para>
219     /// <para></para>
220     /// </param>
221     /// <param name="secondSource">
222     /// <para>The second source.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="secondTarget">
226     /// <para>The second target.</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 FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
235     ↪ TLink secondSource, TLink secondTarget)
236     ↪ => firstSource < secondSource || firstSource == secondSource && firstTarget <
237     ↪ 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">

```

```

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         => firstSource > secondSource || firstSource == secondSource && firstTarget >
268         ↪ secondTarget;
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.LeftAsSource = Zero;
285         link.RightAsSource = Zero;
286         link.SizeAsSource = Zero;
287     }
288 }

```

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>

```

```

38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt64ExternalLinksSourcesSizeBalancedTreeMethods(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.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <returns>
188     /// <para>The link</para>

```

```

186     /// <para></para>
187     /// </returns>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override TLink GetTreeRoot() => Header->RootAsSource;
190
191     /// <summary>
192     /// <para>
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 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>

```



```

262     /// <para></para>
263     /// </returns>
264     [MethodImpl(MethodImplOptions.AggressiveInlining)]
265     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
266         ↪ TLink secondSource, TLink secondTarget)
267         => firstSource > secondSource || firstSource == secondSource && firstTarget >
268             ↪ secondTarget;
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.LeftAsSource = Zero;
285         link.RightAsSource = Zero;
286         link.SizeAsSource = Zero;
287     }
288 }

```

1.69 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethodsBase

```

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>

```

```

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
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     /// <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].RightAsTarget;
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].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) => LinksIndexParts[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>

```

```

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.
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->RootAsTarget;
193
194    /// <summary>
195    /// <para>
196    /// Gets the base part value using the specified node.
197    /// </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].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)
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>

```

```

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.LeftAsTarget = Zero;
283         link.RightAsTarget = Zero;
284         link.SizeAsTarget = Zero;
285     }
286 }
287 }

```

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 :
16     ↪ UInt64ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt64ExternalLinksTargetsSizeBalancedTreeMethods"/>
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 UInt64ExternalLinksTargetsSizeBalancedTreeMethods(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>

```

```

54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLink GetLeftReference(TLink 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 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     /// </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    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLink GetRight(TLink node) => LinksIndexParts[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    /// </param>
118    /// <param name="left">
119    /// <para>The left.</para>
120    /// </param>
121    /// <para></para>
122    /// </param>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override void SetLeft(TLink node, TLink left) =>
125        ↳ 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>

```

```

205 [MethodImpl(MethodImplOptions.AggressiveInlining)]
206 protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].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)
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)]

```



```

279     protected override void ClearNode(TLink node)
280     {
281         ref var link = ref LinksIndexParts[node];
282         link.LeftAsTarget = Zero;
283         link.RightAsTarget = Zero;
284         link.SizeAsTarget = Zero;
285     }
286 }
287 }

```

1.71 ./csharp/Platform.Data.Doublets.Memory.Split.Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase

```

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>
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         /// <para></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
65         ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
66         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
67         ↪ linksIndexParts, LinksHeader<TLink>* header)

```

```

64         : 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     /// </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 > OUL;
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
252     ↪ 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>

```

```

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>

```

```

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 }

```

1.72 ./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 :
        ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
17     {

```

```

18     /// <summary>
19     /// <para>
20     /// The links data parts.
21     /// </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 cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
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 UInt64InternalLinksSizeBalancedTreeMethodsBase(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() => 0UL;
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>

```

```

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
    ↪ 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>

```

```

400     /// <para></para>
401     /// </returns>
402     [MethodImpl(MethodImplOptions.AggressiveInlining)]
403     protected override bool FirstIsToTheLeftOfSecond(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     /// </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 }

```

1.73 ./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     }

```

```

46     /// <summary>
47     /// <para>
48     /// Gets the link data part reference using the specified link.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="link">
53     /// <para>The link.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>A ref raw link data part of t link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
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 }

```

1.74 ./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 :
        ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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>

```

```

38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public
41     ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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     /// <para></para>
51     /// </summary>
52     /// <param name="node">
53     /// <para>The node.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>The ref link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref TLink GetLeftReference(TLink node) => ref
62     ↪ LinksIndexParts[node].LeftAsSource;
63
64     /// <summary>
65     /// <para>
66     /// Gets the right reference using the specified node.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="node">
71     /// <para>The node.</para>
72     /// <para></para>
73     /// </param>
74     /// <returns>
75     /// <para>The ref link</para>
76     /// <para></para>
77     /// </returns>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected override ref TLink GetRightReference(TLink node) => ref
80     ↪ LinksIndexParts[node].RightAsSource;
81
82     /// <summary>
83     /// <para>
84     /// Gets the left using the specified node.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="node">
89     /// <para>The node.</para>
90     /// <para></para>
91     /// </param>
92     /// <returns>
93     /// <para>The link</para>
94     /// <para></para>
95     /// </returns>
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
98
99     /// <summary>
100    /// <para>
101    /// Gets the right using the specified node.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    /// <param name="node">
106    /// <para>The node.</para>
107    /// <para></para>
108    /// </param>
109    /// <returns>
110    /// <para>The link</para>
111    /// <para></para>
112    /// </returns>
113    [MethodImpl(MethodImplOptions.AggressiveInlining)]
114    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;

```

```

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">

```

```

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].RootAsSource;
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].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) =>
        ↪ SearchCore(GetTreeRoot(source), target);
267 }
268 }

```

1.75 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethod

```

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 :
        ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64InternalLinksSourcesSizeBalancedTreeMethods"/>
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="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 UInt64InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, 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
        ↪ 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>

```

```

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>
141    /// <para></para>
142    /// </param>

```

```

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 /// </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 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>

```

```

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 }

```

1.76 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods

```

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 :
16          ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17      {
18          /// <summary>
19          /// <para>
20          /// Initializes a new <see
21          ↪ cref="UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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
41    ↪ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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    /// <para></para>
51    /// </summary>
52    /// <param name="node">
53    /// <para>The node.</para>
54    /// <para></para>
55    /// </param>
56    /// <returns>
57    /// <para>The ref ulong</para>
58    /// <para></para>
59    /// </returns>
60    [MethodImpl(MethodImplOptions.AggressiveInlining)]
61    protected override ref ulong GetLeftReference(ulong node) => ref
62    ↪ LinksIndexParts[node].LeftAsTarget;
63
64    /// <summary>
65    /// <para>
66    /// Gets the right reference using the specified node.
67    /// </para>
68    /// <para></para>
69    /// </summary>
70    /// <param name="node">
71    /// <para>The node.</para>
72    /// <para></para>
73    /// </param>
74    /// <returns>
75    /// <para>The ref ulong</para>
76    /// <para></para>
77    /// </returns>
78    [MethodImpl(MethodImplOptions.AggressiveInlining)]
79    protected override ref ulong GetRightReference(ulong node) => ref
80    ↪ LinksIndexParts[node].RightAsTarget;
81
82    /// <summary>
83    /// <para>
84    /// Gets the left using the specified node.
85    /// </para>
86    /// <para></para>
87    /// </summary>
88    /// <param name="node">
89    /// <para>The node.</para>
90    /// <para></para>
91    /// </param>
92    /// <returns>
93    /// <para>The link</para>
94    /// <para></para>
95    /// </returns>
96    [MethodImpl(MethodImplOptions.AggressiveInlining)]
97    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
98

```

```

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)]
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>

```

```

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    /// </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 }

```

1.77 ./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 :
        ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
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="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 UInt64InternalLinksTargetsSizeBalancedTreeMethods(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 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     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The ref ulong</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected override ref ulong GetRightReference(ulong node) => ref
76     ↪ LinksIndexParts[node].RightAsTarget;
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].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) => LinksIndexParts[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>

```

```

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 }

```

1.78 ./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
45             ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
46
47         /// <summary>
48         /// <para>
49         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> 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         /// <param name="memoryReservationStep">
62         /// <para>A memory reservation step.</para>
63         /// <para></para>
64         /// </param>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
67             ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
68             ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
69             ↪ IndexTreeType.Default, useLinkedList: true) { }
70
71         /// <summary>
72         /// <para>
73         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
74         /// </para>
75         /// <para></para>
76         /// </summary>
77         /// <param name="dataMemory">
78         /// <para>A data memory.</para>
79         /// <para></para>
80         /// </param>
81         /// <param name="indexMemory">
82         /// <para>A index memory.</para>
83         /// <para></para>
84         /// </param>
85         /// <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
138             ↳ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
139             ↳ _linksDataParts, _linksIndexParts, _header);
140     }
141     Init(dataMemory, indexMemory);
142 }
143
144 /// <summary>
145 /// <para>
146 /// Sets the pointers using the specified data memory.
147 /// </para>
148 /// </summary>
149 /// <param name="dataMemory">
150 /// <para>The data memory.</para>
151 /// </param>
152 /// <param name="indexMemory">
153 /// <para>The index memory.</para>
154 /// </param>
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 protected override void SetPointers(IResizableDirectMemory dataMemory,
157     ↳ 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 /// </summary>
183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
184 protected override void ResetPointers()
185 {
186     base.ResetPointers();
187     _linksDataParts = null;
188     _linksIndexParts = null;
189     _header = null;
190 }
191
192 /// <summary>
193 /// <para>
194 /// Gets the header reference.
195 /// </para>
196 /// </summary>
197 /// <returns>
198 /// <para>A ref links header of t link</para>
199 /// </returns>
200 [MethodImpl(MethodImplOptions.AggressiveInlining)]
201 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
202
203 /// <summary>
204 /// <para>
205 /// Gets the link data part reference using the specified link index.
206 /// </para>
207 /// </summary>

```

```

210    /// <param name="linkIndex">
211    /// <para>The link index.</para>
212    /// <para></para>
213    /// </param>
214    /// <returns>
215    /// <para>A ref raw link data part of t link</para>
216    /// <para></para>
217    /// </returns>
218    [MethodImpl(MethodImplOptions.AggressiveInlining)]
219    protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
220    ↪ => ref _linksDataParts[linkIndex];
221
222    /// <summary>
223    /// <para>
224    /// Gets the link index part reference 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 ref raw link index part of t link</para>
234    /// <para></para>
235    /// </returns>
236    [MethodImpl(MethodImplOptions.AggressiveInlining)]
237    protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
238    ↪ linkIndex) => ref _linksIndexParts[linkIndex];
239
240    /// <summary>
241    /// <para>
242    /// Determines whether this instance are equal.
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 AreEqual(ulong first, ulong second) => first == second;
260
261    /// <summary>
262    /// <para>
263    /// Determines whether this instance less than.
264    /// </para>
265    /// <para></para>
266    /// </summary>
267    /// <param name="first">
268    /// <para>The first.</para>
269    /// <para></para>
270    /// </param>
271    /// <param name="second">
272    /// <para>The second.</para>
273    /// <para></para>
274    /// </param>
275    /// <returns>
276    /// <para>The bool</para>
277    /// <para></para>
278    /// </returns>
279    [MethodImpl(MethodImplOptions.AggressiveInlining)]
280    protected override bool LessThan(ulong first, ulong second) => first < second;
281
282    /// <summary>
283    /// <para>
284    /// Determines whether this instance less or equal than.
285    /// </para>
286    /// <para></para>
287    /// </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() => OUL;
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        {

```

```

39         _links = links;
40         _header = header;
41     }
42
43     /// <summary>
44     /// <para>
45     /// Gets the link data part reference using the specified link.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="link">
50     /// <para>The link.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>A ref raw link data part of t link</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
59         ↪ ref _links[link];
60
61     /// <summary>
62     /// <para>
63     /// Gets the header reference.
64     /// </para>
65     /// <para></para>
66     /// </summary>
67     /// <returns>
68     /// <para>A ref links header of t link</para>
69     /// <para></para>
70     /// </returns>
71     [MethodImpl(MethodImplOptions.AggressiveInlining)]
72     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
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>

```

```

36     /// </summary>
37     protected readonly TLink Break;
38     /// <summary>
39     /// <para>
40     /// The continue.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     protected readonly TLink Continue;
45     /// <summary>
46     /// <para>
47     /// The links.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     protected readonly byte* Links;
52     /// <summary>
53     /// <para>
54     /// The header.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     protected readonly byte* Header;
59
60     /// <summary>
61     /// <para>
62     /// Initializes a new <see cref="LinksAvlBalancedTreeMethodsBase"/> instance.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <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,
        ↪ secondLink.Source, secondLink.Target);
250     }
251
252     /// <summary>
253     /// <para>
254     /// Determines whether this instance first is to the right 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 FirstIsToTheRightOfSecond(TLink first, TLink second)
272 {
273     ref var firstLink = ref GetLinkReference(first);
274     ref var secondLink = ref GetLinkReference(second);
275     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
276         ↪ secondLink.Source, secondLink.Target);
277 }
278
279 /// <summary>
280 /// <para>
281 /// Gets the size value using the specified value.
282 /// </para>
283 /// <para></para>
284 /// </summary>
285 /// <param name="value">
286 /// <para>The value.</para>
287 /// <para></para>
288 /// </param>
289 /// <returns>
290 /// <para>The link</para>
291 /// <para></para>
292 /// </returns>
293 [MethodImpl(MethodImplOptions.AggressiveInlining)]
294 protected virtual TLink GetSizeValue(TLink value) => Bit<TLink>.PartialRead(value, 5,
295     ↪ -5);
296
297 /// <summary>
298 /// <para>
299 /// Sets the size value using the specified stored value.
300 /// </para>
301 /// <para></para>
302 /// </summary>
303 /// <param name="storedValue">
304 /// <para>The stored value.</para>
305 /// <para></para>
306 /// </param>
307 /// <param name="size">
308 /// <para>The size.</para>
309 /// <para></para>
310 /// </param>
311 [MethodImpl(MethodImplOptions.AggressiveInlining)]
312 protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
313     ↪ Bit<TLink>.PartialWrite(storedValue, size, 5, -5);
314
315 /// <summary>
316 /// <para>
317 /// Determines whether this instance get left is child value.
318 /// </para>
319 /// <para></para>
320 /// </summary>
321 /// <param name="value">
322 /// <para>The value.</para>
323 /// <para></para>
324 /// </param>
325 /// <returns>
326 /// <para>The bool</para>
327 /// <para></para>
328 /// </returns>
329 [MethodImpl(MethodImplOptions.AggressiveInlining)]
330 protected virtual bool GetLeftIsChildValue(TLink value)
331 {
332     unchecked
333     {
334         return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 4, 1));
335         //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 4, 1), default);
336     }
337 }
338
339 /// <summary>
340 /// <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,
357                 ↪ _boolToAddressConverter.Convert(value), 4, 1);
358             storedValue = modified;
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     /// <summary>
385     /// <para>
386     /// Sets the right is child value using the specified stored value.
387     /// </para>
388     /// <para></para>
389     /// </summary>
390     /// <param name="storedValue">
391     /// <para>The stored value.</para>
392     /// <para></para>
393     /// </param>
394     /// <param name="value">
395     /// <para>The value.</para>
396     /// <para></para>
397     /// </param>
398     [MethodImpl(MethodImplOptions.AggressiveInlining)]
399     protected virtual void SetRightIsChildValue(ref TLink storedValue, bool value)
400     {
401         unchecked
402         {
403             var previousValue = storedValue;
404             var modified = Bit<TLink>.PartialWrite(previousValue,
405                 ↪ _boolToAddressConverter.Convert(value), 3, 1);
406             storedValue = modified;
407         }
408     }
409     /// <summary>
410     /// <para>
411     /// Determines whether this instance is child.
412     /// </para>
413     /// </summary>

```



```

414    /// <para></para>
415    /// </summary>
416    /// <param name="parent">
417    /// <para>The parent.</para>
418    /// <para></para>
419    /// </param>
420    /// <param name="possibleChild">
421    /// <para>The possible child.</para>
422    /// <para></para>
423    /// </param>
424    /// <returns>
425    /// <para>The bool</para>
426    /// <para></para>
427    /// </returns>
428    [MethodImpl(MethodImplOptions.AggressiveInlining)]
429    protected bool IsChild(TLink parent, TLink possibleChild)
430    {
431        var parentSize = GetSize(parent);
432        var childSize = GetSizeOrZero(possibleChild);
433        return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
434    }
435
436    /// <summary>
437    /// <para>
438    /// Gets the balance value using the specified stored value.
439    /// </para>
440    /// <para></para>
441    /// </summary>
442    /// <param name="storedValue">
443    /// <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
463    /// <summary>
464    /// <para>
465    /// Sets the balance value using the specified stored value.
466    /// </para>
467    /// <para></para>
468    /// </summary>
469    /// <param name="storedValue">
470    /// <para>The stored value.</para>
471    /// <para></para>
472    /// </param>
473    /// <param name="value">
474    /// <para>The value.</para>
475    /// <para></para>
476    /// </param>
477    [MethodImpl(MethodImplOptions.AggressiveInlining)]
478    protected virtual void SetBalanceValue(ref TLink storedValue, sbyte value)
479    {
480        unchecked
481        {
482            var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
483                ↪ value & 3);
484            var modified = Bit<TLink>.PartialWrite(storedValue, packagedValue, 0, 3);
485            storedValue = modified;
486        }
487    }
488
489    /// <summary>
490    /// <para>
491    /// The zero.

```

```

489 /// </para>
490 /// <para></para>
491 /// </summary>
492 public TLink this[TLink index]
493 {
494     [MethodImpl(MethodImplOptions.AggressiveInlining)]
495     get
496     {
497         var root = GetTreeRoot();
498         if (GreaterOrEqualThan(index, GetSize(root)))
499         {
500             return Zero;
501         }
502         while (!EqualToZero(root))
503         {
504             var left = GetLeftOrDefault(root);
505             var leftSize = GetSizeOrZero(left);
506             if (LessThan(index, leftSize))
507             {
508                 root = left;
509                 continue;
510             }
511             if (AreEqual(index, leftSize))
512             {
513                 return root;
514             }
515             root = GetRightOrDefault(root);
516             index = Subtract(index, Increment(leftSize));
517         }
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></param>

```

```

563     /// </param>
564     /// <returns>
565     /// <para>The link</para>
566     /// <para></para>
567     /// </returns>
568     [MethodImpl(MethodImplOptions.AggressiveInlining)]
569     public TLink CountUsages(TLink link)
570     {
571         var root = GetTreeRoot();
572         var total = GetSize(root);
573         var totalRightIgnore = Zero;
574         while (!EqualToZero(root))
575         {
576             var @base = GetBasePartValue(root);
577             if (LessOrEqualThan(@base, link))
578             {
579                 root = GetRightOrDefault(root);
580             }
581             else
582             {
583                 totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
584                 root = GetLeftOrDefault(root);
585             }
586         }
587         root = GetTreeRoot();
588         var totalLeftIgnore = Zero;
589         while (!EqualToZero(root))
590         {
591             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                 root = GetRightOrDefault(root);
600             }
601         }
602         return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
603     }
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     {

```

```

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.
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="LinksRecursionlessSizeBalancedTreeMethodsBase"/>
    ↳ 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 LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
    ↳ byte* links, byte* header)
75 {
76     Links = links;
77     Header = header;
78     Break = constants.Break;
79     Continue = constants.Continue;
80 }
81
82 /// <summary>
83 /// <para>
84 /// Gets the tree root.
85 /// </para>
86 /// <para></para>
87 /// </summary>
88 /// <returns>
89 /// <para>The link</para>
90 /// <para></para>
91 /// </returns>
92 [MethodImpl(MethodImplOptions.AggressiveInlining)]
93 protected abstract TLink GetTreeRoot();
94
95 /// <summary>
96 /// <para>
97 /// Gets the base part value using the specified link.
98 /// </para>

```

```

99     /// <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    /// 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    /// 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 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 }

```

1.82 ./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> :
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.
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>
58         protected readonly byte* Header;
59
60         /// <summary>
61         /// <para>
62         /// Initializes a new <see cref="LinksSizeBalancedTreeMethodsBase" /> instance.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <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">

```

```

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     /// <summary>
83     /// <para>
84     /// Gets the tree root.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <returns>
89     /// <para>The link</para>
90     /// <para></para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     protected abstract TLink GetTreeRoot();
94
95     /// <summary>
96     /// <para>
97     /// Gets the base part value using the specified link.
98     /// </para>
99     /// <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    /// 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
140    ↪ rootSource, TLink rootTarget);
141
142    /// <summary>
143    /// <para>
144    /// Determines whether this instance first is to the left of second.
145    /// </para>
146    /// <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 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,
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>
279 /// <para></para>
280 /// </summary>
281 public TLink this[TLink index]
282 {
283     [MethodImpl(MethodImplOptions.AggressiveInlining)]
284     get
285     {
286         var root = GetTreeRoot();
287         if (GreaterOrEqualThan(index, GetSize(root)))
288         {
289             return Zero;
290         }
291         while (!EqualToZero(root))
292         {
293             var left = GetLeftOrDefault(root);
294             var leftSize = GetSizeOrZero(left);
295             if (LessThan(index, leftSize))
296             {

```

```

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
    ↳ broken)
306 }
307 }
308
309 /// <summary>
310 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↳ (концом).
311 /// </summary>
312 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
313 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
314 /// <returns>Индекс искомой связи.</returns>
315 [MethodImpl(MethodImplOptions.AggressiveInlining)]
316 public TLink Search(TLink source, TLink target)
317 {
318     var root = GetTreeRoot();
319     while (!EqualToZero(root))
320     {
321         ref var rootLink = ref GetLinkReference(root);
322         var rootSource = rootLink.Source;
323         var rootTarget = rootLink.Target;
324         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
            ↳ node.Key < root.Key
325         {
326             root = GetLeftOrDefault(root);
327         }
328         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
            ↳ node.Key > root.Key
329         {
330             root = GetRightOrDefault(root);
331         }
332         else // node.Key == root.Key
333         {
334             return root;
335         }
336     }
337     return Zero;
338 }
339
340 // TODO: Return indices range instead of references count
341 /// <summary>
342 /// <para>
343 /// Counts the usages using the specified link.
344 /// </para>
345 /// <para></para>
346 /// </summary>
347 /// <param name="link">
348 /// <para>The link.</para>
349 /// <para></para>
350 /// </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,
    ↳ 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>

```

```

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 link</para>
49     /// <para></para>
50     /// </returns>
51     [MethodImpl(MethodImplOptions.AggressiveInlining)]
52     protected override ref TLink GetLeftReference(TLink node) => ref
53         ↪ GetLinkReference(node).LeftAsSource;
54
55     /// <summary>
56     /// <para>
57     /// Gets the right reference using the specified node.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="node">
62     /// <para>The node.</para>
63     /// <para></para>
64     /// </param>
65     /// <returns>
66     /// <para>The ref link</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref TLink GetRightReference(TLink node) => ref
71         ↪ GetLinkReference(node).RightAsSource;
72
73     /// <summary>
74     /// <para>
75     /// Gets the left using the specified node.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <param name="node">
80     /// <para>The node.</para>
81     /// <para></para>
82     /// </param>
83     /// <returns>
84     /// <para>The link</para>
85     /// <para></para>
86     /// </returns>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;
89
90     /// <summary>
91     /// <para>
92     /// Gets the right using the specified node.
93     /// </para>
94     /// <para></para>
95     /// </summary>
96     /// <param name="node">
97     /// <para>The node.</para>
98     /// <para></para>
99     /// </param>
100    /// <returns>
101    /// <para>The link</para>
102    /// <para></para>
103    /// </returns>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
106
107    /// <summary>
108    /// <para>
109    /// Sets the left using the specified node.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="node">
114    /// <para>The node.</para>
115    /// <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>
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).SizeAsSource);
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>
169     /// <param name="size">
170     /// <para>The size.</para>
171     /// <para></para>
172     /// </param>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override void SetSize(TLink node, TLink size) => SetSizeValue(ref
175         ↪ GetLinkReference(node).SizeAsSource, size);
176
177     /// <summary>
178     /// <para>
179     /// Determines whether this instance get left is child.
180     /// </para>
181     /// <para></para>
182     /// </summary>
183     /// <param name="node">
184     /// <para>The node.</para>
185     /// <para></para>
186     /// </param>
187     /// <returns>
188     /// <para>The bool</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

188     protected override bool GetLeftIsChild(TLink node) =>
189         ↪ GetLeftIsChildValue(GetLinkReference(node).SizeAsSource);
190
191     /// <summary>
192     /// <para>
193     /// Sets the left is child using the specified node.
194     /// </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) =>
206         ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance get right is child.
211     /// </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) =>
223         ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsSource);
224
225     /// <summary>
226     /// <para>
227     /// Sets the right is child using the specified node.
228     /// </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).SizeAsSource, 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     /// <para></para>
250     /// </param>
251     /// <returns>
252     /// <para>The sbyte</para>
253     /// <para></para>
254     /// </returns>
255     [MethodImpl(MethodImplOptions.AggressiveInlining)]
256     protected override sbyte GetBalance(TLink node) =>
257         ↪ GetBalanceValue(GetLinkReference(node).SizeAsSource);
258
259     /// <summary>
260     /// <para>

```

```

260     /// Sets the balance using the specified node.
261     /// </para>
262     /// <para></para>
263     /// </summary>
264     /// <param name="node">
265     /// <para>The node.</para>
266     /// <para></para>
267     /// </param>
268     /// <param name="value">
269     /// <para>The value.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
        ↳ GetLinkReference(node).SizeAsSource, value);

274
275     /// <summary>
276     /// <para>
277     /// Gets the tree root.
278     /// </para>
279     /// <para></para>
280     /// </summary>
281     /// <returns>
282     /// <para>The link</para>
283     /// <para></para>
284     /// </returns>
285     [MethodImpl(MethodImplOptions.AggressiveInlining)]
286     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;

287
288     /// <summary>
289     /// <para>
290     /// 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,
        ↳ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
        ↳ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));

333
334     /// <summary>

```

```

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

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         ref var link = ref GetLinkReference(node);
376         link.LeftAsSource = Zero;
377         link.RightAsSource = Zero;
378         link.SizeAsSource = Zero;
379     }
380 }
381 }
382 }

```

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

```

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> :
        ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="LinksSourcesRecursionlessSizeBalancedTreeMethods"/>
19        ↪ instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <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 LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
36         ↳ byte* links, 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).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 link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref TLink GetRightReference(TLink node) => ref
72         ↳ GetLinkReference(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 link</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;
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>

```

```

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>
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).SizeAsSource;
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)]
173    protected override void SetSize(TLink node, TLink size) =>
174        ↪ GetLinkReference(node).SizeAsSource = size;

```



```

175     /// Gets the tree root.
176     /// </para>
177     /// <para></para>
178     /// </summary>
179     /// <returns>
180     /// <para>The link</para>
181     /// <para></para>
182     /// </returns>
183     [MethodImpl(MethodImplOptions.AggressiveInlining)]
184     protected override TLink GetTreeRoot() => GetHeaderReference().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 link</para>
198     /// <para></para>
199     /// </returns>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(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>
226     /// <para>The bool</para>
227     /// <para></para>
228     /// </returns>
229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
230     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
231     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
232     ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, 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">

```

```

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 }

```

1.85 ./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> :
        ↪ LinksSizeBalancedTreeMethodsBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="LinksSourcesSizeBalancedTreeMethods" /> 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 LinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
            ↪ byte* 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 link</para>
49     /// <para></para>
50     /// </returns>
51     [MethodImpl(MethodImplOptions.AggressiveInlining)]
52     protected override ref TLink GetLeftReference(TLink node) => ref
53     ↪ GetLinkReference(node).LeftAsSource;
54
55     /// <summary>
56     /// <para>
57     /// Gets the right reference using the specified node.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="node">
62     /// <para>The node.</para>
63     /// <para></para>
64     /// </param>
65     /// <returns>
66     /// <para>The ref link</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref TLink GetRightReference(TLink node) => ref
71     ↪ GetLinkReference(node).RightAsSource;
72
73     /// <summary>
74     /// <para>
75     /// Gets the left using the specified node.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <param name="node">
80     /// <para>The node.</para>
81     /// <para></para>
82     /// </param>
83     /// <returns>
84     /// <para>The link</para>
85     /// <para></para>
86     /// </returns>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;
89
90     /// <summary>
91     /// <para>
92     /// Gets the right using the specified node.
93     /// </para>
94     /// <para></para>
95     /// </summary>
96     /// <param name="node">
97     /// <para>The node.</para>
98     /// <para></para>
99     /// </param>
100    /// <returns>
101    /// <para>The link</para>
102    /// <para></para>
103    /// </returns>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
106
107    /// <summary>
108    /// <para>
109    /// Sets the left using the specified node.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="node">
114    /// <para>The node.</para>
115    /// <para></para>
116    /// </param>
117    /// <param name="left">
118    /// <para>The left.</para>
119    /// <para></para>
120    /// </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) => 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) =>
    ↳ GetLinkReference(node).SizeAsSource = size;
172
173 /// <summary>
174 /// <para>
175 /// Gets the tree root.
176 /// </para>
177 /// <para></para>
178 /// </summary>
179 /// <returns>
180 /// <para>The link</para>
181 /// <para></para>
182 /// </returns>
183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
184 protected override TLink GetTreeRoot() => GetHeaderReference().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 link</para>
198    /// <para></para>
199    /// </returns>
200    [MethodImpl(MethodImplOptions.AggressiveInlining)]
201    protected override TLink GetBasePartValue(TLink link) => GetLinkReference(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>
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 }
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> :
15         ↳ LinksAvlBalancedTreeMethodsBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksTargetsAvlBalancedTreeMethods" /> 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 LinksTargetsAvlBalancedTreeMethods(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).LeftAsTarget;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </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).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 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) =>
    ↪ GetLinkReference(node).LeftAsTarget = 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).RightAsTarget = 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) =>
    ↳ GetSizeValue(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) => 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) =>
223        ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsTarget);
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).SizeAsTarget, 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).SizeAsTarget);
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).SizeAsTarget, value);
278
279    /// <summary>
280    /// <para>
281    /// Gets the tree root.
282    /// </para>
283    /// <para></para>
284    /// </summary>
285    /// <returns>

```

```

282    /// <para>The link</para>
283    /// <para></para>
284    /// </returns>
285    [MethodImpl(MethodImplOptions.AggressiveInlining)]
286    protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
287
288    /// <summary>
289    /// <para>
290    /// 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).Target;
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(firstTarget, secondTarget) ||
334    ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
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>

```

```

358     /// <para></para>
359     /// </returns>
360     [MethodImpl(MethodImplOptions.AggressiveInlining)]
361     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
362
363     /// <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.LeftAsTarget = Zero;
378         link.RightAsTarget = Zero;
379         link.SizeAsTarget = Zero;
380     }
381 }
382 }

```

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

```

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 recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class LinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
        ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="LinksTargetsRecursionlessSizeBalancedTreeMethods"/>
19        ↪ 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 LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
            ↪ byte* links, 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>

```

```

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

```

```

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).RightAsTarget = 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).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">
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).SizeAsTarget = 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().RootAsTarget;
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>

```

```

198     /// <para></para>
199     /// </returns>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Target;
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>
226     /// <para>The bool</para>
227     /// <para></para>
228     /// </returns>
229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
230     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
231     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
232     ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, 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(TLink firstSource, TLink firstTarget,
262     ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
263     ↪ (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>

```

```

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> :
15        ↳ LinksSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="LinksTargetsSizeBalancedTreeMethods"/> 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 LinksTargetsSizeBalancedTreeMethods(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).LeftAsTarget;
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>

```

```

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).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 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) =>
    ↪ GetLinkReference(node).LeftAsTarget = 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).RightAsTarget = 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) => 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>

```

```

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(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 }

```

1.89 ./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 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 UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
47         ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
48         ↪ memoryReservationStep) { }
49
50         /// <summary>
51         /// <para>
52         /// Initializes a new <see cref="UnitedMemoryLinks" /> instance.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <param name="memory">
57         /// <para>A memory.</para>
58         /// <para></para>
59         /// </param>
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         public UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
62         ↪ DefaultLinksSizeStep) { }
63
64         /// <summary>
65         /// <para>
66         /// Initializes a new <see cref="UnitedMemoryLinks" /> instance.
67         /// </para>
68         /// <para></para>
69         /// </summary>
70         /// <param name="memory">
71         /// <para>A memory.</para>
72         /// <para></para>
73         /// </param>
74         /// <param name="memoryReservationStep">
75         /// <para>A memory reservation step.</para>
76         /// <para></para>
77         /// </param>
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep) :
80         ↪ this(memory, memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
81         ↪ IndexTreeType.Default) { }
82
83         /// <summary>
84         /// <para>
85         /// 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,
100     ↪ LinksConstants<TLink> constants, IndexTreeType indexTreeType) : base(memory,
101     ↪ memoryReservationStep, constants)
102     {
103         if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
104         {
105             _createSourceTreeMethods = () => new
106             ↪ LinksSourcesAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
107             _createTargetTreeMethods = () => new
108             ↪ LinksTargetsAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
109         }
110         else
111         {
112             _createSourceTreeMethods = () => new
113             ↪ LinksSourcesSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
114             _createTargetTreeMethods = () => new
115             ↪ LinksTargetsSizeBalancedTreeMethods<TLink>(Constants, _links, _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         _links = (byte*)memory.Pointer;
134         _header = _links;
135         SourcesTreeMethods = _createSourceTreeMethods();
136         TargetsTreeMethods = _createTargetTreeMethods();
137         UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_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.

```

```

151     /// </para>
152     /// <para></para>
153     /// </summary>
154     /// <returns>
155     /// <para>A ref links header of t link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪ AsRef<LinksHeader<TLink>>(_header);
160
161     /// <summary>
162     /// <para>
163     /// Gets the link reference using the specified link index.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="linkIndex">
168     /// <para>The link index.</para>
169     /// <para></para>
170     /// </param>
171     /// <returns>
172     /// <para>A ref raw link of t link</para>
173     /// <para></para>
174     /// </returns>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override ref RawLink<TLink> GetLinkReference(TLink linkIndex) => ref
        ↪ AsRef<RawLink<TLink>>(_links + (LinkSizeInBytes * ConvertToInt64(linkIndex)));
177 }
178 }

```

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 =
            ↪ EqualityComparer<TLink>.Default;
27         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
28         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
            ↪ UncheckedConverter<TLink, long>.Default;
29         private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
            ↪ UncheckedConverter<long, TLink>.Default;
30         private static readonly TLink _zero = default;
31         private static readonly TLink _one = Arithmetic.Increment(_zero);
32
33         /// <summary>Возвращает размер одной связи в байтах.</summary>
34         /// <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 LinkSizeInBytes = RawLink<TLink>.SizeInBytes;
39
40         /// <summary>
41         /// <para>
42         /// The size in bytes.
43         /// </para>
44         /// <para></para>

```

```

45     /// </summary>
46     public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
47
48     /// <summary>
49     /// <para>
50     /// The link size in bytes.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
55
56     /// <summary>
57     /// <para>
58     /// The memory.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     protected readonly IResizableDirectMemory _memory;
63     /// <summary>
64     /// <para>
65     /// The memory reservation step.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     protected readonly long _memoryReservationStep;
70
71     /// <summary>
72     /// <para>
73     /// The targets tree methods.
74     /// </para>
75     /// <para></para>
76     /// </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
96     /// <summary>
97     /// Возвращает общее число связей находящихся в хранилище.
98     /// </summary>
99     protected virtual TLink Total
100     {
101         [MethodImpl(MethodImplOptions.AggressiveInlining)]
102         get
103         {
104             ref var header = ref GetHeaderReference();
105             return Subtract(header.AllocatedLinks, header.FreeLinks);
106         }
107     }
108
109     /// <summary>
110     /// <para>
111     /// Gets the constants value.
112     /// </para>
113     /// <para></para>
114     /// </summary>
115     public virtual LinksConstants<TLink> Constants
116     {
117         [MethodImpl(MethodImplOptions.AggressiveInlining)]
118         get;
119     }
120
121     /// <summary>
122     /// <para>
123     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.

```

```

122     /// </para>
123     /// <para></para>
124     /// </summary>
125     /// <param name="memory">
126     /// <para>A memory.</para>
127     /// <para></para>
128     /// </param>
129     /// <param name="memoryReservationStep">
130     /// <para>A memory reservation step.</para>
131     /// <para></para>
132     /// </param>
133     /// <param name="constants">
134     /// <para>A constants.</para>
135     /// <para></para>
136     /// </param>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
        ↳ memoryReservationStep, LinksConstants<TLink> constants)
139     {
140         _memory = memory;
141         memoryReservationStep = memoryReservationStep;
142         Constants = constants;
143     }
144
145     /// <summary>
146     /// <para>
147     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <param name="memory">
152     /// <para>A memory.</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(IList<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         }
274     }
275 }

```



```

271     else if (AreEqual(target, any))
272     {
273         return SourcesTreeMethods.CountUsages(source);
274     }
275     else //if(source != Any && target != Any)
276     {
277         // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
278         var link = SourcesTreeMethods.Search(source, target);
279         return AreEqual(link, constants.Null) ? GetZero() : GetOne();
280     }
281 }
282 else
283 {
284     if (!Exists(index))
285     {
286         return GetZero();
287     }
288     if (AreEqual(source, any) && AreEqual(target, any))
289     {
290         return GetOne();
291     }
292     ref var storedLinkValue = ref GetLinkReference(index);
293     if (!AreEqual(source, any) && !AreEqual(target, any))
294     {
295         if (AreEqual(storedLinkValue.Source, source) &&
296             ⇨ AreEqual(storedLinkValue.Target, target))
297         {
298             return GetOne();
299         }
300         return GetZero();
301     }
302     var value = default(TLink);
303     if (AreEqual(source, any))
304     {
305         value = target;
306     }
307     if (AreEqual(target, any))
308     {
309         value = source;
310     }
311     if (AreEqual(storedLinkValue.Source, value) ||
312         ⇨ AreEqual(storedLinkValue.Target, value))
313     {
314         return GetOne();
315     }
316     return GetZero();
317 }
318 }
319
320 throw new NotSupportedException("Другие размеры и способы ограничений не
321 ⇨ поддерживаются.");
322
323 /// <summary>
324 /// <para>
325 /// Eaches the handler.
326 /// </para>
327 /// <para></para>
328 /// </summary>
329 /// <param name="handler">
330 /// <para>The handler.</para>
331 /// <para></para>
332 /// </param>
333 /// <param name="restriction">
334 /// <para>The substitution.</para>
335 /// <para></para>
336 /// </param>
337 /// <exception cref="NotSupportedException">
338 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
339 /// <para></para>
340 /// </exception>
341 /// <returns>
342 /// <para>The link</para>
343 /// <para></para>
344 /// </returns>
345 [MethodImpl(MethodImplOptions.AggressiveInlining)]
346 public virtual TLink Each(ICollection<TLink> restriction, ReadHandler<TLink> handler)
347 {
348     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))
378     {
379         if (AreEqual(value, any))
380         {
381             return Each(Array.Empty<TLink>(), handler);
382         }
383         if (AreEqual(Each(new Link<TLink>(index, value, any), handler), @break))
384         {
385             return @break;
386         }
387         return Each(new Link<TLink>(index, any, value), handler);
388     }
389     else
390     {
391         if (!Exists(index))
392         {
393             return @continue;
394         }
395         if (AreEqual(value, any))
396         {
397             return handler(GetLinkStruct(index));
398         }
399         ref var storedLinkValue = ref GetLinkReference(index);
400         if (AreEqual(storedLinkValue.Source, value) ||
401             AreEqual(storedLinkValue.Target, value))
402         {
403             return handler(GetLinkStruct(index));
404         }
405         return @continue;
406     }
407 }
408 if (restriction.Count == 3)
409 {
410     var source = restriction[constants.SourcePart];
411     var target = restriction[constants.TargetPart];
412     if (AreEqual(index, any))
413     {
414         if (AreEqual(source, any) && AreEqual(target, any))
415         {
416             return Each(Array.Empty<TLink>(), handler);
417         }
418         else if (AreEqual(source, any))
419         {
420             return TargetsTreeMethods.EachUsage(target, handler);
421         }
422         else if (AreEqual(target, any))
423         {

```

```

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 :
            ↳ handler(GetLinkStruct(link));
429     }
430 }
431 else
432 {
433     if (!Exists(index))
434     {
435         return @continue;
436     }
437     if (AreEqual(source, any) && AreEqual(target, any))
438     {
439         return handler(GetLinkStruct(index));
440     }
441     ref var storedLinkValue = ref GetLinkReference(index);
442     if (!AreEqual(source, any) && !AreEqual(target, any))
443     {
444         if (AreEqual(storedLinkValue.Source, source) &&
445             AreEqual(storedLinkValue.Target, target))
446         {
447             return handler(GetLinkStruct(index));
448         }
449         return @continue;
450     }
451     var value = default(TLink);
452     if (AreEqual(source, any))
453     {
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(IList<TLink> restriction, IList<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
509   ↳ пространство
510 /// </remarks>
511 [MethodImpl(MethodImplOptions.AggressiveInlining)]
512 public virtual TLink Create(IList<TLink> substitution, WriteHandler<TLink> handler)
513 {
514     ref var header = ref GetHeaderReference();
515     var freeLink = header.FirstFreeLink;
516     if (!AreEqual(freeLink, Constants.Null))
517     {
518         UnusedLinksListMethods.Detach(freeLink);
519     }
520     else
521     {
522         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
523         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
524         {
525             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
526         }
527         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
528         {
529             _memory.ReservedCapacity += _memory.ReservationStep;
530             SetPointers(_memory);
531             header = ref GetHeaderReference();
532             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
533                 ↳ LinkSizeInBytes);
534         }
535         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
536         _memory.UsedCapacity += LinkSizeInBytes;
537     }
538     return handler != null ? handler(null, new Link<TLink>(freeLink, Constants.Null,
539         ↳ Constants.Null)) : Constants.Continue;
540 }
541
542 /// <summary>
543 /// <para>
544 /// Deletes the substitution.
545 /// </para>
546 /// <para></para>
547 /// </summary>
548 /// <param name="restriction">
549 /// <para>The substitution.</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         {

```

```

568         UnusedLinksListMethods.Detach(header.AllocatedLinks);
569         header.AllocatedLinks = Decrement(header.AllocatedLinks);
570         _memory.UsedCapacity -= LinkSizeInBytes;
571     }
572     return handler != null ? handler(before, null) : Constants.Continue;
573 }
574 return Constants.Continue;
575 }
576
577 /// <summary>
578 /// <para>
579 /// Gets the link struct using the specified link index.
580 /// </para>
581 /// <para></para>
582 /// </summary>
583 /// <param name="linkIndex">
584 /// <para>The link index.</para>
585 /// <para></para>
586 /// </param>
587 /// <returns>
588 /// <para>A list of t link</para>
589 /// <para></para>
590 /// </returns>
591 [MethodImpl(MethodImplOptions.AggressiveInlining)]
592 public IList<TLink> GetLinkStruct(TLink linkIndex)
593 {
594     ref var link = ref GetLinkReference(linkIndex);
595     return new Link<TLink>(linkIndex, link.Source, link.Target);
596 }
597
598 /// <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 /// <para></para>
614 /// </summary>
615 [MethodImpl(MethodImplOptions.AggressiveInlining)]
616 protected virtual void ResetPointers()
617 {
618     SourcesTreeMethods = null;
619     TargetsTreeMethods = null;
620     UnusedLinksListMethods = null;
621 }
622
623 /// <summary>
624 /// <para>
625 /// Gets the header reference.
626 /// </para>
627 /// <para></para>
628 /// </summary>
629 /// <returns>
630 /// <para>A ref links header of t link</para>
631 /// <para></para>
632 /// </returns>
633 [MethodImpl(MethodImplOptions.AggressiveInlining)]
634 protected abstract ref LinksHeader<TLink> GetHeaderReference();
635
636 /// <summary>
637 /// <para>
638 /// Gets the link reference using the specified link index.
639 /// </para>
640 /// <para></para>
641 /// </summary>
642 /// <param name="linkIndex">
643 /// <para>The link index.</para>
644 /// <para></para>
645 /// </param>
646 /// <returns>

```

```

646 /// <para>A ref raw link of t link</para>
647 /// <para></para>
648 /// </returns>
649 [MethodImpl(MethodImplOptions.AggressiveInlining)]
650 protected abstract ref RawLink<TLink> GetLinkReference(TLink linkIndex);
651
652 /// <summary>
653 /// <para>
654 /// Determines whether this instance exists.
655 /// </para>
656 /// <para></para>
657 /// </summary>
658 /// <param name="link">
659 /// <para>The link.</para>
660 /// <para></para>
661 /// </param>
662 /// <returns>
663 /// <para>The bool</para>
664 /// <para></para>
665 /// </returns>
666 [MethodImpl(MethodImplOptions.AggressiveInlining)]
667 protected virtual bool Exists(TLink link)
668     => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
669     && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
670     && !IsUnusedLink(link);
671
672 /// <summary>
673 /// <para>
674 /// Determines whether this instance is unused link.
675 /// </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
701 /// <summary>
702 /// <para>
703 /// Gets the one.
704 /// </para>
705 /// <para></para>
706 /// </summary>
707 /// <returns>
708 /// <para>The link</para>
709 /// <para></para>
710 /// </returns>
711 [MethodImpl(MethodImplOptions.AggressiveInlining)]
712 protected virtual TLink GetOne() => _one;
713
714 /// <summary>
715 /// <para>
716 /// Gets the zero.
717 /// </para>
718 /// <para></para>
719 /// </summary>
720 /// <returns>
721 /// <para>The link</para>
722 /// <para></para>
723 /// </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>
753 /// </summary>
754 /// <param name="first">
755 /// <para>The first.</para>
756 /// <para></para>
757 /// </param>
758 /// <param name="second">
759 /// <para>The second.</para>
760 /// <para></para>
761 /// </param>
762 /// <returns>
763 /// <para>The bool</para>
764 /// <para></para>
765 /// </returns>
766 [MethodImpl(MethodImplOptions.AggressiveInlining)]
767 protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
768     ↪ second) < 0;
769
770 /// <summary>
771 /// <para>
772 /// Determines whether this instance less or equal than.
773 /// </para>
774 /// <para></para>
775 /// </summary>
776 /// <param name="first">
777 /// <para>The first.</para>
778 /// <para></para>
779 /// </param>
780 /// <param name="second">
781 /// <para>The second.</para>
782 /// <para></para>
783 /// </param>
784 /// <returns>
785 /// <para>The bool</para>
786 /// <para></para>
787 /// </returns>
788 [MethodImpl(MethodImplOptions.AggressiveInlining)]
789 protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
790     ↪ _comparer.Compare(first, second) <= 0;
791
792 /// <summary>
793 /// <para>
794 /// Determines whether this instance greater than.
795 /// </para>
796 /// <para></para>
797 /// </summary>
798 /// <param name="first">
799 /// <para>The first.</para>
800 /// <para></para>
801 /// </param>
802 /// <param name="second">
803 /// <para>The second.</para>
804 /// <para></para>
805 /// </param>
806 /// <returns>
807 /// <para>The bool</para>
808 /// <para></para>
809 /// </returns>
810 [MethodImpl(MethodImplOptions.AggressiveInlining)]
811 protected virtual bool GreaterThan(TLink first, TLink second) =>
812     ↪ _comparer.Compare(first, second) > 0;
813
814 /// <summary>
815 /// <para>
816 /// Determines whether this instance greater or equal than.
817 /// </para>
818 /// <para></para>
819 /// </summary>
820 /// <param name="first">
821 /// <para>The first.</para>
822 /// <para></para>
823 /// </param>
824 /// <param name="second">
825 /// <para>The second.</para>
826 /// <para></para>
827 /// </param>
828 /// <returns>
829 /// <para>The bool</para>
830 /// <para></para>
831 /// </returns>
832 [MethodImpl(MethodImplOptions.AggressiveInlining)]
833 protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
834     ↪ _comparer.Compare(first, second) >= 0;

```

```

798     /// </param>
799     /// <param name="second">
800     /// <para>The second.</para>
801     /// <para></para>
802     /// </param>
803     /// <returns>
804     /// <para>The bool</para>
805     /// <para></para>
806     /// </returns>
807     [MethodImpl(MethodImplOptions.AggressiveInlining)]
808     protected virtual bool GreaterThan(TLink first, TLink second) =>
809         ↪ _comparer.Compare(first, second) > 0;
810
811     /// <summary>
812     /// <para>
813     /// Determines whether this instance greater or equal than.
814     /// </para>
815     /// <para></para>
816     /// </summary>
817     /// <param name="first">
818     /// <para>The first.</para>
819     /// <para></para>
820     /// </param>
821     /// <param name="second">
822     /// <para>The second.</para>
823     /// <para></para>
824     /// </param>
825     /// <returns>
826     /// <para>The bool</para>
827     /// <para></para>
828     /// </returns>
829     [MethodImpl(MethodImplOptions.AggressiveInlining)]
830     protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
831         ↪ _comparer.Compare(first, second) >= 0;
832
833     /// <summary>
834     /// <para>
835     /// Converts the to int 64 using the specified value.
836     /// </para>
837     /// <para></para>
838     /// </summary>
839     /// <param name="value">
840     /// <para>The value.</para>
841     /// <para></para>
842     /// </param>
843     /// <returns>
844     /// <para>The long</para>
845     /// <para></para>
846     /// </returns>
847     [MethodImpl(MethodImplOptions.AggressiveInlining)]
848     protected virtual long ConvertToInt64(TLink value) =>
849         ↪ _addressToInt64Converter.Convert(value);
850
851     /// <summary>
852     /// <para>
853     /// Converts the to address using the specified value.
854     /// </para>
855     /// <para></para>
856     /// </summary>
857     /// <param name="value">
858     /// <para>The value.</para>
859     /// <para></para>
860     /// </param>
861     /// <returns>
862     /// <para>The link</para>
863     /// <para></para>
864     /// </returns>
865     [MethodImpl(MethodImplOptions.AggressiveInlining)]
866     protected virtual TLink ConvertToAddress(long value) =>
867         ↪ _int64ToAddressConverter.Convert(value);
868
869     /// <summary>
870     /// <para>
871     /// Adds the first.
872     /// </para>
873     /// <para></para>
874     /// </summary>
875     /// <param name="first">

```



```

872     /// <para>The first.</para>
873     /// <para></para>
874     /// </param>
875     /// <param name="second">
876     /// <para>The second.</para>
877     /// <para></para>
878     /// </param>
879     /// <returns>
880     /// <para>The link</para>
881     /// <para></para>
882     /// </returns>
883     [MethodImpl(MethodImplOptions.AggressiveInlining)]
884     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
885     ↪ second);
886
887     /// <summary>
888     /// <para>
889     /// Subtracts the first.
890     /// </para>
891     /// <para></para>
892     /// </summary>
893     /// <param name="first">
894     /// <para>The first.</para>
895     /// <para></para>
896     /// </param>
897     /// <param name="second">
898     /// <para>The second.</para>
899     /// <para></para>
900     /// </param>
901     /// <returns>
902     /// <para>The link</para>
903     /// <para></para>
904     /// </returns>
905     [MethodImpl(MethodImplOptions.AggressiveInlining)]
906     protected virtual TLink Subtract(TLink first, TLink second) =>
907     ↪ Arithmetic<TLink>.Subtract(first, second);
908
909     /// <summary>
910     /// <para>
911     /// Increments the link.
912     /// </para>
913     /// <para></para>
914     /// </summary>
915     /// <param name="link">
916     /// <para>The link.</para>
917     /// <para></para>
918     /// </param>
919     /// <returns>
920     /// <para>The link</para>
921     /// <para></para>
922     /// </returns>
923     [MethodImpl(MethodImplOptions.AggressiveInlining)]
924     protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
925
926     /// <summary>
927     /// <para>
928     /// Decrements the link.
929     /// </para>
930     /// <para></para>
931     /// </summary>
932     /// <param name="link">
933     /// <para>The link.</para>
934     /// <para></para>
935     /// </param>
936     /// <returns>
937     /// <para>The link</para>
938     /// <para></para>
939     /// </returns>
940     [MethodImpl(MethodImplOptions.AggressiveInlining)]
941     protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
942
943     #region Disposable
944
945     /// <summary>
946     /// <para>
947     /// Gets the allow multiple dispose calls value.
948     /// </para>
949     /// <para></para>

```

```

948     /// </summary>
949     protected override bool AllowMultipleDisposeCalls
950     {
951         [MethodImpl(MethodImplOptions.AggressiveInlining)]
952         get => true;
953     }
954
955     /// <summary>
956     /// <para>
957     /// Disposes the manual.
958     /// </para>
959     /// <para></para>
960     /// </summary>
961     /// <param name="manual">
962     /// <para>The manual.</para>
963     /// <para></para>
964     /// </param>
965     /// <param name="wasDisposed">
966     /// <para>The was disposed.</para>
967     /// <para></para>
968     /// </param>
969     [MethodImpl(MethodImplOptions.AggressiveInlining)]
970     protected override void Dispose(bool manual, bool wasDisposed)
971     {
972         if (!wasDisposed)
973         {
974             ResetPointers();
975             _memory.DisposeIfPossible();
976         }
977     }
978
979     #endregion
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         /// <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;

```

```

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 /// <para></para>
54 /// </returns>
55 [MethodImpl(MethodImplOptions.AggressiveInlining)]
56 protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪ 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 /// <para></para>
67 /// </param>
68 /// <returns>
69 /// <para>A ref raw link of t link</para>
70 /// <para></para>
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
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 =
160         ↪ element;
161
162     /// <summary>
163     /// <para>
164     /// Sets the last using the specified element.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="element">
169     /// <para>The element.</para>
170     /// <para></para>
171     /// </param>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
174         ↪ element;
175
176     /// <summary>
177     /// <para>
178     /// Sets the previous using the specified element.
179     /// </para>
180     /// <para></para>
181     /// </summary>
182     /// <param name="element">
183     /// <para>The element.</para>
184     /// <para></para>
185     /// </param>
186     /// <param name="previous">
187     /// <para>The previous.</para>
188     /// <para></para>
189     /// </param>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override void SetPrevious(TLink element, TLink previous) =>
192         ↪ GetLinkReference(element).Source = previous;
193
194     /// <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         ↪ GetLinkReference(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 }

```

1.92 ./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>

```

```

46     /// <para></para>
47     /// </summary>
48     public TLink LeftAsSource;
49     /// <summary>
50     /// <para>
51     /// The right as source.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     public TLink RightAsSource;
56     /// <summary>
57     /// <para>
58     /// The size as source.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLink SizeAsSource;
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 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,
139     ↪ SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
140
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static bool operator ==(RawLink<TLink> left, RawLink<TLink> right) =>
143     ↪ left.Equals(right);
144
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static bool operator !=(RawLink<TLink> left, RawLink<TLink> right) => !(left ==
    ↪ right);
    }
}

```

1.93 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethods

```

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
26         /// <summary>
27         /// <para>
28         /// The header.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly LinksHeader<uint>* Header;
33
34         /// <summary>
35         /// <para>
36         /// Initializes a new <see cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
37         ↪ instance.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="constants">
42         /// <para>A constants.</para>
43         /// <para></para>
44         /// </param>
45         /// <param name="links">
46         /// <para>A links.</para>
47         /// <para></para>
48         /// </param>
49         /// <param name="header">
50         /// <para>A header.</para>
51         /// <para></para>
52         /// </param>
53     }
54 }

```

```

48     /// <para></para>
49     /// </param>
50     protected UInt32LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<uint>
    ↪     constants, RawLink<uint>* links, LinksHeader<uint>* header)
51         : base(constants, (byte*)links, (byte*)header)
52     {
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 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>
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
    ↪ always >= 0 for uint

```

```

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
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>

```

```

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 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,
        ↪ secondLink.Source, secondLink.Target);
360 }
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,
        ↪ 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,
    ↳ RawLink<uint>* links, LinksHeader<uint>* header)
    : base(constants, (byte*)links, (byte*)header)
51      {
52          Links = links;
53          Header = header;
54      }
55
56      /// <summary>
57      /// <para>
58      /// Gets the zero.
59      /// </para>
60      /// <para></para>
61      /// </summary>
62      /// <returns>
63      /// <para>The uint</para>
64      /// <para></para>
65      /// </returns>
66      [MethodImpl(MethodImplOptions.AggressiveInlining)]
67      protected override uint GetZero() => 0U;
68
69      /// <summary>
70      /// <para>
71      /// Determines whether this instance equal to zero.
72      /// </para>
73      /// <para></para>
74      /// </summary>
75      /// <param name="value">
76      /// <para>The value.</para>
77      /// <para></para>
78      /// </param>
79      /// <returns>
80      /// <para>The bool</para>
81      /// <para></para>
82      /// </returns>
83      [MethodImpl(MethodImplOptions.AggressiveInlining)]
84      protected override bool EqualToZero(uint value) => value == 0U;
85
86      /// <summary>
87

```

```

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     /// <summary>
167     /// <para>
168     /// Determines whether this instance greater or equal than zero.
169     /// </para>
170     /// <para></para>
171     /// </summary>
172     /// <param name="value">
173     /// <para>The value.</para>
174     /// <para></para>
175     /// </param>
176     /// <returns>
177     /// <para>The bool</para>
178     /// <para></para>
179     /// </returns>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     protected override bool GreaterOrEqualThanZero(uint value) => true; // value >= 0 is
182     ↪ always true for uint
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(uint value) => value == 0U; // value is
200     ↪ always >= 0 for uint
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(uint first, uint 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(uint value) => false; // value < 0 is always false
239     ↪ for uint
240
241     /// <summary>
242     /// <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 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    {
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>

```

```

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>

```

```

49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
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
        ↪ Links[node].RightAsSource;
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].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     /// <summary>
138     /// <para>
139     /// Gets the size using the specified node.
140     /// </para>
141     /// <para></para>
142     /// </summary>
143     /// <param name="node">
144     /// <para>The node.</para>
145     /// <para></para>
146     /// </param>
147     /// <returns>
148     /// <para>The uint</para>
149     /// <para></para>
150     /// </returns>
151     [MethodImpl(MethodImplOptions.AggressiveInlining)]
152     protected override uint GetSize(uint node) => Links[node].SizeAsSource;

153     /// <summary>
154     /// <para>
155     /// Sets the size using the specified node.
156     /// </para>
157     /// <para></para>
158     /// </summary>
159     /// <param name="node">
160     /// <para>The node.</para>
161     /// <para></para>
162     /// </param>
163     /// <param name="size">
164     /// <para>The size.</para>
165     /// <para></para>
166     /// </param>
167     [MethodImpl(MethodImplOptions.AggressiveInlining)]
168     protected override void SetSize(uint node, uint size) => Links[node].SizeAsSource = size;

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

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

197     /// <summary>
198     /// <para>
199     /// Gets the base part value using the specified link.
200     /// </para>
201     /// <para></para>
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>
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     => 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(uint node)
278     {
279         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 :
15         ↳ UInt32LinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32LinksSourcesSizeBalancedTreeMethods"/> 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 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;

```

```

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].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
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>
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     => 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(uint node)
278     {
279         ref var link = ref Links[node];
280         link.LeftAsSource = 0U;
281         link.RightAsSource = 0U;
282         link.SizeAsSource = 0U;
283     }
284 }

```

1.97 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTree

```

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
    ↳ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
    ↳ constants, RawLink<uint>* links, LinksHeader<uint>* 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 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
    ↳ 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 =
137        ↪ 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 uint</para>
151    /// <para></para>
152    /// </returns>
153    [MethodImpl(MethodImplOptions.AggressiveInlining)]
154    protected override uint GetSize(uint node) => Links[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     /// </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     /// <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         => 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     /// <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 :
        ↪ UInt32LinksSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> 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 UInt32LinksTargetsSizeBalancedTreeMethods(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].LeftAsTarget;
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].RightAsTarget;
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].LeftAsTarget;
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>
99     /// <returns>
100    /// <para>The uint</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected override uint GetRight(uint node) => Links[node].RightAsTarget;
105
106    /// <summary>
107    /// <para>
108    /// Sets the left using the specified node.
109    /// </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)

```



```

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     /// <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 }

```

1.99 ./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 реализацию 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         /// <para></para>
30         /// </summary>
31         /// <param name="address">
32         /// <para>A address.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt32UnitedMemoryLinks(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 UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
47         ↳ FileMappedResizableDirectMemory(address, memoryReservationStep),
48         ↳ memoryReservationStep) { }
49
50         /// <summary>
51         /// <para>
52         /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
53         /// </para>
54         /// <para></para>
55     }
56 }

```

```

48     /// </summary>
49     /// <param name="memory">
50     /// <para>A memory.</para>
51     /// <para></para>
52     /// </param>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
        ↳ DefaultLinksSizeStep) { }
55
56     /// <summary>
57     /// <para>
58     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="memory">
63     /// <para>A memory.</para>
64     /// <para></para>
65     /// </param>
66     /// <param name="memoryReservationStep">
67     /// <para>A memory reservation step.</para>
68     /// <para></para>
69     /// </param>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep) : this(memory, memoryReservationStep,
        ↳ Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }
72
73     /// <summary>
74     /// <para>
75     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <param name="memory">
80     /// <para>A memory.</para>
81     /// <para></para>
82     /// </param>
83     /// <param name="memoryReservationStep">
84     /// <para>A memory reservation step.</para>
85     /// <para></para>
86     /// </param>
87     /// <param name="constants">
88     /// <para>A constants.</para>
89     /// <para></para>
90     /// </param>
91     /// <param name="indexTreeType">
92     /// <para>A index tree type.</para>
93     /// <para></para>
94     /// </param>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep, LinksConstants<uint> constants, IndexTreeType indexTreeType)
        ↳ : base(memory, memoryReservationStep, constants)
97     {
98         if (indexTreeType == IndexTreeType.SizeBalancedTree)
99         {
100             _createSourceTreeMethods = () => new
                ↳ UInt32LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
101             _createTargetTreeMethods = () => new
                ↳ UInt32LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
102         }
103         else
104         {
105             _createSourceTreeMethods = () => new
                ↳ UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
                ↳ _header);
106             _createTargetTreeMethods = () => new
                ↳ UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
                ↳ _header);
107         }
108         Init(memory, memoryReservationStep);
109     }
110
111     /// <summary>
112     /// <para>
113     /// Sets the pointers using the specified memory.

```

```

114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="memory">
118    /// <para>The memory.</para>
119    /// <para></para>
120    /// </param>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    protected override void SetPointers(IResizableDirectMemory memory)
123    {
124        _header = (LinksHeader<uint>*)memory.Pointer;
125        _links = (RawLink<uint>*)memory.Pointer;
126        SourcesTreeMethods = _createSourceTreeMethods();
127        TargetsTreeMethods = _createTargetTreeMethods();
128        UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_links, _header);
129    }
130
131    /// <summary>
132    /// <para>
133    /// Resets the pointers.
134    /// </para>
135    /// <para></para>
136    /// </summary>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    protected override void ResetPointers()
139    {
140        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
174    ↪ _links[linkIndex];
175
176    /// <summary>
177    /// <para>
178    /// Determines whether this instance are equal.
179    /// </para>
180    /// <para></para>
181    /// </summary>
182    /// <param name="first">
183    /// <para>The first.</para>
184    /// <para></para>
185    /// </param>
186    /// <param name="second">
187    /// <para>The second.</para>
188    /// <para></para>
189    /// </param>
190    /// <returns>
191    /// <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 :
    ↳ LinksAvlBalancedTreeMethodsBase<ulong>
17  {
18      /// <summary>
19      /// <para>
20      /// The links.
21      /// </para>
22      /// <para></para>
23      /// </summary>
24      protected new readonly RawLink<ulong>* Links;
25      /// <summary>
26      /// <para>
27      /// The header.
28      /// </para>
29      /// <para></para>
30      /// </summary>
31      protected new readonly LinksHeader<ulong>* Header;
32
33      /// <summary>
34      /// <para>
35      /// Initializes a new <see cref="UInt64LinksAvlBalancedTreeMethodsBase"/> instance.
36      /// </para>
37      /// <para></para>
38      /// </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,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header)
    : base(constants, (byte*)links, (byte*)header)
52      {
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() => 0UL;
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 == 0UL;
86
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 FirstIsToLeftOfSecond(ulong first, ulong second)
357    {
358        ref var firstLink = ref Links[first];
359        ref var secondLink = ref Links[second];
360        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
361            ↪ secondLink.Source, secondLink.Target);
362    }
363
364    /// <summary>
365    /// <para>
366    /// Determines whether this instance first is to the right of second.
367    /// </para>
368    /// <para></para>
369    /// </summary>
370    /// <param name="first">
371    /// <para>The first.</para>
372    /// <para></para>
373    /// </param>
374    /// <param name="second">
375    /// <para>The second.</para>
376    /// <para></para>
377    /// </param>
378    /// <returns>
379    /// <para>The bool</para>
380    /// <para></para>
381    /// </returns>
382    [MethodImpl(MethodImplOptions.AggressiveInlining)]
383    protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
384    {
385        ref var firstLink = ref Links[first];
386        ref var secondLink = ref Links[second];
387        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
388            ↪ secondLink.Source, secondLink.Target);
389    }
390
391    /// <summary>
392    /// <para>
393    /// Gets the size value using the specified value.
394    /// </para>
395    /// <para></para>
396    /// </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;
423
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance get left is child value.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="value">
431     /// <para>The value.</para>
432     /// <para></para>
433     /// </param>
434     /// <returns>
435     /// <para>The bool</para>
436     /// <para></para>
437     /// </returns>
438     [MethodImpl(MethodImplOptions.AggressiveInlining)]
439     protected override bool GetLeftIsChildValue(ulong value) => (value & 16UL) >> 4 == 1UL;
440
441     /// <summary>
442     /// <para>
443     /// Sets the left is child value using the specified stored value.
444     /// </para>
445     /// <para></para>
446     /// </summary>
447     /// <param name="storedValue">
448     /// <para>The stored value.</para>
449     /// <para></para>
450     /// </param>
451     /// <param name="value">
452     /// <para>The value.</para>
453     /// <para></para>
454     /// </param>
455     [MethodImpl(MethodImplOptions.AggressiveInlining)]
456     protected override void SetLeftIsChildValue(ref ulong storedValue, bool value) =>
457         ↪ storedValue = storedValue & 4294967279UL | (As<bool, byte>(ref value) & 1UL) << 4;
458
459     /// <summary>
460     /// <para>
461     /// Determines whether this instance get right is child value.
462     /// </para>
463     /// <para></para>
464     /// </summary>
465     /// <param name="value">
466     /// <para>The value.</para>
467     /// <para></para>
468     /// </param>
469     /// <returns>
470     /// <para>The bool</para>
471     /// <para></para>
472     /// </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">
499 /// <para>The value.</para>
500 /// <para></para>
501 /// </param>
502 /// <returns>
503 /// <para>The sbyte</para>
504 /// <para></para>
505 /// </returns>
506 [MethodImpl(MethodImplOptions.AggressiveInlining)]
507 protected override sbyte GetBalanceValue(ulong value) => unchecked((sbyte)(value & 7UL |
508     ↪ 0xF8UL * ((value & 4UL) >> 2))); // if negative, then continue ones to the end of
509     ↪ sbyte
510
511 /// <summary>
512 /// <para>
513 /// Sets the balance value using the specified stored value.
514 /// </para>
515 /// <para></para>
516 /// </summary>
517 /// <param name="storedValue">
518 /// <para>The stored value.</para>
519 /// <para></para>
520 /// </param>
521 /// <param name="value">
522 /// <para>The value.</para>
523 /// <para></para>
524 /// </param>
525 [MethodImpl(MethodImplOptions.AggressiveInlining)]
526 protected override void SetBalanceValue(ref ulong storedValue, sbyte value) =>
527     ↪ storedValue = unchecked(storedValue & 4294967288UL | (ulong)((byte)value >> 5 & 4 |
528     ↪ value & 3) & 7UL);
529
530 /// <summary>
531 /// <para>
532 /// Gets the header reference.
533 /// </para>
534 /// <para></para>
535 /// </summary>
536 /// <returns>
537 /// <para>A ref links header of ulong</para>
538 /// <para></para>
539 /// </returns>
540 [MethodImpl(MethodImplOptions.AggressiveInlining)]
541 protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
542
543 /// <summary>
544 /// <para>
545 /// Gets the link reference using the specified link.
546 /// </para>
547 /// <para></para>
548 /// </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 :
16     ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<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="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase" />
37         ↪ instance.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="constants">
42         /// <para>A constants.</para>
43         /// <para></para>
44         /// </param>
45         /// <param name="links">
46         /// <para>A links.</para>
47         /// <para></para>
48         /// </param>
49         /// <param name="header">
50         /// <para>A header.</para>
51         /// <para></para>
52         /// </param>
53         protected UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
54     ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
55         : base(constants, (byte*)links, (byte*)header)
56     {
57         Links = links;
58         Header = header;
59     }
60
61     /// <summary>
62     /// <para>
63     /// Gets the zero.
64     /// </para>
65     /// <para></para>
66     /// </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
    ↪ 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
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">
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     /// <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 FirstIsToLeftOfSecond(ulong first, ulong 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
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>

```

```

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(ulong first, ulong 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 ulong</para>
396     /// <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         /// <summary>
26         /// <para>
27         /// 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="UInt64LinksSizeBalancedTreeMethodsBase"/> 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 UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
51     ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
52     : base(constants, (byte*)links, (byte*)header)
53     {
54         Links = links;
55         Header = header;
56     }
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>

```

```

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>
399    /// </returns>
400    [MethodImpl(MethodImplOptions.AggressiveInlining)]
401    protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
402
403    /// <summary>
404    /// <para>
405    /// Gets the link reference using the specified link.
406    /// </para>
407    /// <para></para>
408    /// </summary>
409    /// <param name="link">
410    /// <para>The link.</para>
411    /// <para></para>
412    /// </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 }

```

1.104 ./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     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksSourcesAvlBalancedTreeMethods :
15         ↳ UInt64LinksAvlBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64LinksSourcesAvlBalancedTreeMethods"/> 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 UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
36             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37             ↳ { }
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 ulong</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref ulong GetLeftReference(ulong node) => ref
55             ↳ 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 ulong</para>
69         /// <para></para>
70         /// </returns>

```

```

65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
        ↳ Links[node].RightAsSource;
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].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) => GetSizeValue(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) => SetSizeValue(ref
    ↪ Links[node].SizeAsSource, 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) =>
    ↪ GetLeftIsChildValue(Links[node].SizeAsSource);
188
189    //[MethodImpl(MethodImplOptions.AggressiveInlining)]
190    //protected override bool GetLeftIsChild(ulong node) => IsChild(node, GetLeft(node));
191
192    /// <summary>
193    /// <para>
194    /// Sets the left is child 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    /// <param name="value">
203    /// <para>The value.</para>
204    /// <para></para>
205    /// </param>
206    [MethodImpl(MethodImplOptions.AggressiveInlining)]
207    protected override void SetLeftIsChild(ulong node, bool value) =>
    ↪ SetLeftIsChildValue(ref Links[node].SizeAsSource, value);
208
209    /// <summary>
210    /// <para>
211    /// Determines whether this instance get right is child.
212    /// </para>
213    /// <para></para>
214    /// </summary>

```

```

215 /// <param name="node">
216 /// <para>The node.</para>
217 /// <para></para>
218 /// </param>
219 /// <returns>
220 /// <para>The bool</para>
221 /// <para></para>
222 /// </returns>
223 [MethodImpl(MethodImplOptions.AggressiveInlining)]
224 protected override bool GetRightIsChild(ulong node) =>
225     ↪ GetRightIsChildValue(Links[node].SizeAsSource);
226
227 ///[MethodImpl(MethodImplOptions.AggressiveInlining)]
228 ///protected override bool GetRightIsChild(ulong node) => IsChild(node, GetRight(node));
229
230 /// <summary>
231 /// <para>
232 /// Sets the right is child 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 /// <param name="value">
241 /// <para>The value.</para>
242 /// <para></para>
243 /// </param>
244 [MethodImpl(MethodImplOptions.AggressiveInlining)]
245 protected override void SetRightIsChild(ulong node, bool value) =>
246     ↪ SetRightIsChildValue(ref Links[node].SizeAsSource, value);
247
248 /// <summary>
249 /// <para>
250 /// Gets the balance using the specified node.
251 /// </para>
252 /// <para></para>
253 /// </summary>
254 /// <param name="node">
255 /// <para>The node.</para>
256 /// <para></para>
257 /// </param>
258 /// <returns>
259 /// <para>The sbyte</para>
260 /// <para></para>
261 /// </returns>
262 [MethodImpl(MethodImplOptions.AggressiveInlining)]
263 protected override sbyte GetBalance(ulong node) =>
264     ↪ GetBalanceValue(Links[node].SizeAsSource);
265
266 /// <summary>
267 /// <para>
268 /// Sets the balance 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 /// <param name="value">
277 /// <para>The value.</para>
278 /// <para></para>
279 /// </param>
280 [MethodImpl(MethodImplOptions.AggressiveInlining)]
281 protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
282     ↪ Links[node].SizeAsSource, value);
283
284 /// <summary>
285 /// <para>
286 /// Gets the tree root.
287 /// </para>
288 /// <para></para>
289 /// </summary>
290 /// <returns>
291 /// <para>The ulong</para>
292 /// <para></para>

```

```

    /// </returns>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override ulong GetTreeRoot() => Header->RootAsSource;

    /// <summary>
    /// <para>
    /// Gets the base part value using the specified link.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <param name="link">
    /// <para>The link.</para>
    /// <para></para>
    /// </param>
    /// <returns>
    /// <para>The ulong</para>
    /// <para></para>
    /// </returns>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override ulong GetBasePartValue(ulong link) => Links[link].Source;

    /// <summary>
    /// <para>
    /// Determines whether this instance first is to the left of second.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <param name="firstSource">
    /// <para>The first source.</para>
    /// <para></para>
    /// </param>
    /// <param name="firstTarget">
    /// <para>The first target.</para>
    /// <para></para>
    /// </param>
    /// <param name="secondSource">
    /// <para>The second source.</para>
    /// <para></para>
    /// </param>
    /// <param name="secondTarget">
    /// <para>The second target.</para>
    /// <para></para>
    /// </param>
    /// <returns>
    /// <para>The bool</para>
    /// <para></para>
    /// </returns>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
    ↪   ulong secondSource, ulong secondTarget)
    ↪   => firstSource < secondSource || (firstSource == secondSource && firstTarget <
    ↪   secondTarget);

    /// <summary>
    /// <para>
    /// Determines whether this instance first is to the right of second.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <param name="firstSource">
    /// <para>The first source.</para>
    /// <para></para>
    /// </param>
    /// <param name="firstTarget">
    /// <para>The first target.</para>
    /// <para></para>
    /// </param>
    /// <param name="secondSource">
    /// <para>The second source.</para>
    /// <para></para>
    /// </param>
    /// <param name="secondTarget">
    /// <para>The second target.</para>
    /// <para></para>
    /// </param>
    /// <returns>
    /// <para>The bool</para>
    /// <para></para>
    /// </returns>

```

```

365     /// </returns>
366     [MethodImpl(MethodImplOptions.AggressiveInlining)]
367     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
368     ↪     ulong secondSource, ulong secondTarget)
369     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
370     ↪     secondTarget);
371
372     /// <summary>
373     /// <para>
374     /// Clears the node using the specified node.
375     /// </para>
376     /// <para></para>
377     /// </summary>
378     /// <param name="node">
379     /// <para>The node.</para>
380     /// <para></para>
381     /// </param>
382     [MethodImpl(MethodImplOptions.AggressiveInlining)]
383     protected override void ClearNode(ulong node)
384     {
385         ref var link = ref Links[node];
386         link.LeftAsSource = OUL;
387         link.RightAsSource = OUL;
388         link.SizeAsSource = OUL;
389     }
390 }

```

1.105 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods

```

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>

```

```

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
52     ↪ 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 ulong</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ref ulong GetRightReference(ulong 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 ulong</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected override ulong GetLeft(ulong 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>
99     /// <returns>
100    /// <para>The ulong</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
105
106    /// <summary>
107    /// <para>
108    /// Sets the left using the specified node.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <param name="node">
113    /// <para>The node.</para>
114    /// <para></para>
115    /// </param>
116    /// <param name="left">
117    /// <para>The left.</para>
118    /// <para></para>
119    /// </param>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsSource =
122    ↪ left;

```



```

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     /// <summary>
138     /// <para>
139     /// Gets the size using the specified node.
140     /// </para>
141     /// <para></para>
142     /// </summary>
143     /// <param name="node">
144     /// <para>The node.</para>
145     /// <para></para>
146     /// </param>
147     /// <returns>
148     /// <para>The ulong</para>
149     /// <para></para>
150     /// </returns>
151     [MethodImpl(MethodImplOptions.AggressiveInlining)]
152     protected override ulong GetSize(ulong node) => Links[node].SizeAsSource;

153     /// <summary>
154     /// <para>
155     /// Sets the size using the specified node.
156     /// </para>
157     /// <para></para>
158     /// </summary>
159     /// <param name="node">
160     /// <para>The node.</para>
161     /// <para></para>
162     /// </param>
163     /// <param name="size">
164     /// <para>The size.</para>
165     /// <para></para>
166     /// </param>
167     [MethodImpl(MethodImplOptions.AggressiveInlining)]
168     protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsSource =
169         ↳ size;

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

182     /// <summary>
183     /// <para>
184     /// Gets the base part value using the specified link.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="link">
189     /// <para>The link.</para>
190     /// <para></para>
191     /// </param>
192     /// <returns>
193     /// <para>The ulong</para>
194     /// <para></para>
195     /// </returns>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override ulong GetBasePartValue(ulong link) => Links[link].Source;

/// <summary>
/// <para>
/// Determines whether this instance first is to the left of second.
/// </para>
/// <para></para>
/// </summary>
/// <param name="firstSource">
/// <para>The first source.</para>
/// <para></para>
/// </param>
/// <param name="firstTarget">
/// <para>The first target.</para>
/// <para></para>
/// </param>
/// <param name="secondSource">
/// <para>The second source.</para>
/// <para></para>
/// </param>
/// <param name="secondTarget">
/// <para>The second target.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The bool</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
↪   ulong secondSource, ulong secondTarget)
    => firstSource < secondSource || (firstSource == secondSource && firstTarget <
↪   secondTarget);

/// <summary>
/// <para>
/// Determines whether this instance first is to the right of second.
/// </para>
/// <para></para>
/// </summary>
/// <param name="firstSource">
/// <para>The first source.</para>
/// <para></para>
/// </param>
/// <param name="firstTarget">
/// <para>The first target.</para>
/// <para></para>
/// </param>
/// <param name="secondSource">
/// <para>The second source.</para>
/// <para></para>
/// </param>
/// <param name="secondTarget">
/// <para>The second target.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The bool</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
↪   ulong secondSource, ulong secondTarget)
    => firstSource > secondSource || (firstSource == secondSource && firstTarget >
↪   secondTarget);

/// <summary>
/// <para>
/// Clears the node using the specified node.
/// </para>
/// <para></para>
/// </summary>
/// <param name="node">
/// <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 }

```

1.106 ./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
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt64LinksSourcesSizeBalancedTreeMethods"/> 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 UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
36            ↪ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37        { }
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 ulong</para>
51        /// <para></para>
52        /// </returns>
53        [MethodImpl(MethodImplOptions.AggressiveInlining)]
54        protected override ref ulong GetLeftReference(ulong node) => ref
55            ↪ 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 ulong</para>
69        /// <para></para>
70        /// </returns>

```

```

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].RightAsSource;
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].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 =
137         ↪ right;
138
139     /// <summary>
140     /// <para>
141     /// Gets the size using the specified node.
142     /// </para>
143     /// </summary>
144     /// <param name="node">
145     /// <para>The node.</para>
146     /// </para>
147     /// </param>
148     /// <returns>
149     /// <para>The ulong</para>
150     /// </returns>
151     [MethodImpl(MethodImplOptions.AggressiveInlining)]
152     protected override ulong GetSize(ulong node) => Links[node].SizeAsSource;
153
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>
163     /// </param>
164     /// <param name="size">
165     /// <para>The size.</para>
166     /// </para>
167     /// </param>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsSource =
170         ↪ 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     /// </returns>
181     [MethodImpl(MethodImplOptions.AggressiveInlining)]
182     protected override ulong 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>
193     /// </param>
194     /// <returns>
195     /// <para>The ulong</para>
196     /// </returns>
197     [MethodImpl(MethodImplOptions.AggressiveInlining)]
198     protected override ulong GetBasePartValue(ulong link) => Links[link].Source;
199
200     /// <summary>
201     /// <para>
202     /// Determines whether this instance first is to the left of second.
203     /// </para>
204     /// <para></para>
205     /// </summary>
206     /// <param name="firstSource">
207     /// <para>The first source.</para>
208     /// </para>
209     /// </param>
210
211

```

```

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,
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 }

```

1.107 ./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 :
15     ↪ UInt64LinksAvlBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64LinksTargetsAvlBalancedTreeMethods"/> 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 UInt64LinksTargetsAvlBalancedTreeMethods(LinksConstants<ulong> constants,
36         ↪ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37         ↪ { }
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 ulong</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref ulong GetLeftReference(ulong node) => ref
55         ↪ Links[node].LeftAsTarget;
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 ulong</para>
69         /// <para></para>
70         /// </returns>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         protected override ref ulong GetRightReference(ulong node) => ref
73         ↪ Links[node].RightAsTarget;
74
75         /// <summary>
76         /// <para>
77         /// Gets the left using the specified node.
78         /// </para>
79         /// <para></para>
80         /// </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) =>
    ↳ GetLeftIsChildValue(Links[node].SizeAsTarget);
188
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) =>
    ↳ SetLeftIsChildValue(ref Links[node].SizeAsTarget, value);
205
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) =>
    ↳ GetRightIsChildValue(Links[node].SizeAsTarget);
222
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
241    /// <summary>
242    /// <para>
243    /// Gets the balance using the specified node.
244    /// </para>
245    /// <para></para>
246    /// </summary>
247    /// <param name="node">
248    /// <para>The node.</para>
249    /// <para></para>
250    /// </param>
251    /// <returns>
252    /// <para>The sbyte</para>
253    /// <para></para>
254    /// </returns>
255    [MethodImpl(MethodImplOptions.AggressiveInlining)]
256    protected override sbyte GetBalance(ulong node) =>
257        ↪ GetBalanceValue(Links[node].SizeAsTarget);
258
259    /// <summary>
260    /// <para>
261    /// Sets the balance using the specified node.
262    /// </para>
263    /// <para></para>
264    /// </summary>
265    /// <param name="node">
266    /// <para>The node.</para>
267    /// <para></para>
268    /// </param>
269    /// <param name="value">
270    /// <para>The value.</para>
271    /// <para></para>
272    /// </param>
273    [MethodImpl(MethodImplOptions.AggressiveInlining)]
274    protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
275        ↪ Links[node].SizeAsTarget, value);
276
277    /// <summary>
278    /// <para>
279    /// Gets the tree root.
280    /// </para>
281    /// <para></para>
282    /// </summary>
283    /// <returns>
284    /// <para>The ulong</para>
285    /// <para></para>
286    /// </returns>
287    [MethodImpl(MethodImplOptions.AggressiveInlining)]
288    protected override ulong GetTreeRoot() => Header->RootAsTarget;
289
290    /// <summary>
291    /// <para>
292    /// Gets the base part value using the specified link.
293    /// </para>
294    /// <para></para>
295    /// </summary>
296    /// <param name="link">
297    /// <para>The link.</para>
298    /// <para></para>
299    /// </param>
300    /// <returns>
301    /// <para>The ulong</para>
302    /// <para></para>
303    /// </returns>

```

```

301 [MethodImpl(MethodImplOptions.AggressiveInlining)]
302 protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
303
304 /// <summary>
305 /// <para>
306 /// Determines whether this instance first is to the left of second.
307 /// </para>
308 /// <para></para>
309 /// </summary>
310 /// <param name="firstSource">
311 /// <para>The first source.</para>
312 /// <para></para>
313 /// </param>
314 /// <param name="firstTarget">
315 /// <para>The first target.</para>
316 /// <para></para>
317 /// </param>
318 /// <param name="secondSource">
319 /// <para>The second source.</para>
320 /// <para></para>
321 /// </param>
322 /// <param name="secondTarget">
323 /// <para>The second target.</para>
324 /// <para></para>
325 /// </param>
326 /// <returns>
327 /// <para>The bool</para>
328 /// <para></para>
329 /// </returns>
330 [MethodImpl(MethodImplOptions.AggressiveInlining)]
331 protected override bool FirstIsToLeftOfSecond(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 /// <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(ulong firstSource, ulong firstTarget,
364     ↪ ulong secondSource, ulong secondTarget)
365     => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
366     ↪ secondSource);
367
368 /// <summary>
369 /// <para>
370 /// Clears the node using the specified node.
371 /// </para>
372 /// <para></para>
373 /// </summary>
374 /// <param name="node">
375 /// <para>The node.</para>
376 /// <para></para>
377 /// </param>

```

```

374     [MethodImpl(MethodImplOptions.AggressiveInlining)]
375     protected override void ClearNode(ulong node)
376     {
377         ref var link = ref Links[node];
378         link.LeftAsTarget = OUL;
379         link.RightAsTarget = OUL;
380         link.SizeAsTarget = OUL;
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     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods :
15         ↪ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(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].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 ulong</para>
70         /// <para></para>
71         /// </returns>

```

```

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 =
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->RootAsTarget;
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].Target;
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>

```

```

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>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
262     ↪     ulong secondSource, ulong 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(ulong node)
278     {
279         ref var link = ref Links[node];
280         link.LeftAsTarget = OUL;
281         link.RightAsTarget = OUL;
282         link.SizeAsTarget = OUL;
283     }
284 }

```

1.109 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.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 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></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 UInt64LinksTargetsSizeBalancedTreeMethods(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].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 ulong</para>
68        /// <para></para>
69        /// </returns>
70        [MethodImpl(MethodImplOptions.AggressiveInlining)]
71        protected override ref ulong GetRightReference(ulong node) => ref
72            ↳ Links[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 ref ulong</para>
86        /// <para></para>
87        /// </returns>
88        [MethodImpl(MethodImplOptions.AggressiveInlining)]
89        protected override ref ulong GetLeftUsingNode(ulong node) => ref
90            ↳ Links[node].LeftUsingNode;
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 ref ulong</para>
104       /// <para></para>
105       /// </returns>
106       [MethodImpl(MethodImplOptions.AggressiveInlining)]
107       protected override ref ulong GetRightUsingNode(ulong node) => ref
108           ↳ Links[node].RightUsingNode;
109    }
110 }

```



```

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     /// <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(ulong node, ulong size) => Links[node].SizeAsTarget =
170     ↪ 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,
    ↪ 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>
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         => 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     /// <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.LeftAsTarget = OUL;
277         link.RightAsTarget = OUL;
278         link.SizeAsTarget = OUL;
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
           ↳ organizing the storage of links with addresses represented as <see cref="ulong"
           ↳ />.</para>
13     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
           ↳ размером, для организации хранения связей с адресами представленными в виде <see
           ↳ cref="ulong"/>.</para>
14     /// </summary>
15     public unsafe class UInt64UnitedMemoryLinks : UnitedMemoryLinksBase<ulong>
16     {
17         private readonly Func<ILinksTreeMethods<ulong>> _createSourceTreeMethods;
18         private readonly Func<ILinksTreeMethods<ulong>> _createTargetTreeMethods;

```

```

19 private LinksHeader<ulong>* _header;
20 private RawLink<ulong>* _links;
21
22 /// <summary>
23 /// <para>
24 /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
25 /// </para>
26 /// <para></para>
27 /// </summary>
28 /// <param name="address">
29 /// <para>A address.</para>
30 /// <para></para>
31 /// </param>
32 [MethodImpl(MethodImplOptions.AggressiveInlining)]
33 public UInt64UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
34
35 /// <summary>
36 /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
37   ↳ минимальным шагом расширения базы данных.
38 /// </summary>
39 /// <param name="address">Полный путь к файлу базы данных.</param>
40 /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
41   ↳ байтах.</param>
42 [MethodImpl(MethodImplOptions.AggressiveInlining)]
43 public UInt64UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
44   ↳ FileMappedResizableDirectMemory(address, memoryReservationStep),
45   ↳ memoryReservationStep) { }
46
47 /// <summary>
48 /// <para>
49 /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
50 /// </para>
51 /// <para></para>
52 /// </summary>
53 /// <param name="memory">
54 /// <para>A memory.</para>
55 /// <para></para>
56 /// </param>
57 [MethodImpl(MethodImplOptions.AggressiveInlining)]
58 public UInt64UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
59   ↳ DefaultLinksSizeStep) { }
60
61 /// <summary>
62 /// <para>
63 /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
64 /// </para>
65 /// <para></para>
66 /// </summary>
67 /// <param name="memory">
68 /// <para>A memory.</para>
69 /// <para></para>
70 /// </param>
71 /// <param name="memoryReservationStep">
72 /// <para>A memory reservation step.</para>
73 /// <para></para>
74 /// </param>
75 [MethodImpl(MethodImplOptions.AggressiveInlining)]
76 public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
77   ↳ memoryReservationStep) : this(memory, memoryReservationStep,
78   ↳ Default<LinksConstants<ulong>>.Instance, IndexTreeType.Default) { }
79
80 /// <summary>
81 /// <para>
82 /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
83 /// </para>
84 /// <para></para>
85 /// </summary>
86 /// <param name="memory">
87 /// <para>A memory.</para>
88 /// <para></para>
89 /// </param>
90 /// <param name="memoryReservationStep">
91 /// <para>A memory reservation step.</para>
92 /// <para></para>
93 /// </param>
94 /// <param name="constants">
95 /// <para>A constants.</para>
96 /// <para></para>
97 /// </param>

```

```

90     /// </param>
91     /// <param name="indexTreeType">
92     /// <para>A index tree type.</para>
93     /// <para></para>
94     /// </param>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
    ↪ memoryReservationStep, LinksConstants<ulong> constants, IndexTreeType indexTreeType)
    ↪ : base(memory, memoryReservationStep, constants)
97     {
98         if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
99         {
100             _createSourceTreeMethods = () => new
    ↪ UInt64LinksSourcesAvlBalancedTreeMethods(Constants, _links, _header);
101             _createTargetTreeMethods = () => new
    ↪ UInt64LinksTargetsAvlBalancedTreeMethods(Constants, _links, _header);
102         }
103         else if (indexTreeType == IndexTreeType.SizeBalancedTree)
104         {
105             _createSourceTreeMethods = () => new
    ↪ UInt64LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
106             _createTargetTreeMethods = () => new
    ↪ UInt64LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
107         }
108         else
109         {
110             _createSourceTreeMethods = () => new
    ↪ UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
    ↪ _header);
111             _createTargetTreeMethods = () => new
    ↪ UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(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     /// <para></para>
121     /// </summary>
122     /// <param name="memory">
123     /// <para>The memory.</para>
124     /// <para></para>
125     /// </param>
126     [MethodImpl(MethodImplOptions.AggressiveInlining)]
127     protected override void SetPointers(IResizableDirectMemory memory)
128     {
129         _header = (LinksHeader<ulong>*)memory.Pointer;
130         _links = (RawLink<ulong>*)memory.Pointer;
131         SourcesTreeMethods = _createSourceTreeMethods();
132         TargetsTreeMethods = _createTargetTreeMethods();
133         UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
134     }
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();
146         _links = null;
147         _header = null;
148     }
149
150     /// <summary>
151     /// <para>
152     /// Gets the header reference.
153     /// </para>
154     /// <para></para>
155     /// </summary>
156     /// <returns>

```

```

157    /// <para>A ref links header of ulong</para>
158    /// <para></para>
159    /// </returns>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
161    protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
162
163    /// <summary>
164    /// <para>
165    /// Gets the link reference using the specified link index.
166    /// </para>
167    /// <para></para>
168    /// </summary>
169    /// <param name="linkIndex">
170    /// <para>The link index.</para>
171    /// <para></para>
172    /// </param>
173    /// <returns>
174    /// <para>A ref raw link of ulong</para>
175    /// <para></para>
176    /// </returns>
177    [MethodImpl(MethodImplOptions.AggressiveInlining)]
178    protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
179        ↪ _links[linkIndex];
180
181    /// <summary>
182    /// <para>
183    /// Determines whether this instance are equal.
184    /// </para>
185    /// <para></para>
186    /// </summary>
187    /// <param name="first">
188    /// <para>The first.</para>
189    /// <para></para>
190    /// </param>
191    /// <param name="second">
192    /// <para>The second.</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 AreEqual(ulong first, ulong second) => first == second;
201
202    /// <summary>
203    /// <para>
204    /// Determines whether this instance less 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 LessThan(ulong first, ulong second) => first < second;
222
223    /// <summary>
224    /// <para>
225    /// Determines whether this instance less or equal than.
226    /// </para>
227    /// <para></para>
228    /// </summary>
229    /// <param name="first">
230    /// <para>The first.</para>
231    /// <para></para>
232    /// </param>
233    /// <param name="second">
234    /// <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>

```

```

49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public TLink GetValue(TLink @object, TLink property)
54     {
55         var links = _links;
56         var objectProperty = links.SearchOrDefault(@object, property);
57         if (_equalityComparer.Equals(objectProperty, default))
58         {
59             return default;
60         }
61         var constants = links.Constants;
62         var any = constants.Any;
63         var query = new Link<TLink>(any, objectProperty, any);
64         var valueLink = links.SingleOrDefault(query);
65         if (valueLink == null)
66         {
67             return default;
68         }
69         return links.GetTarget(valueLink[constants.IndexPart]);
70     }
71
72     /// <summary>
73     /// <para>
74     /// Sets the value using the specified object.
75     /// </para>
76     /// <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.

```

```

26    /// </para>
27    /// <para></para>
28    /// </summary>
29    /// <param name="links">
30    /// <para>A links.</para>
31    /// <para></para>
32    /// </param>
33    /// <param name="propertyMarker">
34    /// <para>A property marker.</para>
35    /// <para></para>
36    /// </param>
37    /// <param name="propertyValueMarker">
38    /// <para>A property value marker.</para>
39    /// <para></para>
40    /// </param>
41    [MethodImpl(MethodImplOptions.AggressiveInlining)]
42    public PropertyOperator(ILinks<TLink> links, TLink propertyMarker, TLink
    ↪ propertyValueMarker) : base(links)
43    {
44        _propertyMarker = propertyMarker;
45        _propertyValueMarker = propertyValueMarker;
46    }
47
48    /// <summary>
49    /// <para>
50    /// Gets the 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>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)
    ↪ ? default : _links.GetTarget(container);
97
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         /// <para></para>
45         /// </param>
46         /// <param name="stack">
47         /// <para>A stack.</para>
48         /// <para></para>
49         /// </param>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

50 public Stack(ILinks<TLink> links, TLink stack) : base(links) => _stack = stack;
51 [MethodImpl(MethodImplOptions.AggressiveInlining)]
52 private TLink GetStackMarker() => _links.GetSource(_stack);
53 [MethodImpl(MethodImplOptions.AggressiveInlining)]
54 private TLink GetTop() => _links.GetTarget(_stack);
55
56 /// <summary>
57 /// <para>
58 /// Peeks this instance.
59 /// </para>
60 /// <para></para>
61 /// </summary>
62 /// <returns>
63 /// <para>The link</para>
64 /// <para></para>
65 /// </returns>
66 [MethodImpl(MethodImplOptions.AggressiveInlining)]
67 public TLink Peek() => _links.GetTarget(GetTop());
68
69 /// <summary>
70 /// <para>
71 /// Pops this instance.
72 /// </para>
73 /// <para></para>
74 /// </summary>
75 /// <returns>
76 /// <para>The element.</para>
77 /// <para></para>
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
79         ↳ ReaderWriterLockSynchronization(), links) { }
80
81     /// <summary>
82     /// <para>
83     /// Initializes a new <see cref="SynchronizedLinks"/> instance.
84     /// </para>
85     /// <para></para>
86     /// </summary>
87     /// <param name="synchronization">
88     /// <para>A synchronization.</para>
89     /// <para></para>
90     /// </param>
91     /// <param name="links">
92     /// <para>A links.</para>
93     /// <para></para>
94     /// </param>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public SynchronizedLinks(ISynchronization synchronization, ILinks<TLinkAddress> links)
97     {
98         SyncRoot = synchronization;
99         Sync = this;
100         Unsync = links;
101         Constants = links.Constants;
102     }
103
104     /// <summary>
105     /// <para>
106     /// Counts the restriction.
107     /// </para>
108     /// <para></para>
109     /// </summary>
110     /// <param name="restriction">
111     /// <para>The restriction.</para>
112     /// <para></para>
113     /// </param>
114     /// <returns>
115     /// <para>The link address</para>
116     /// <para></para>
117     /// </returns>
118     [MethodImpl(MethodImplOptions.AggressiveInlining)]
119     public TLinkAddress Count(ICollection<TLinkAddress> restriction) =>
120         ↳ SyncRoot.ExecuteReadOperation(restriction, Unsync.Count);
121
122     /// <summary>
123     /// <para>
124     /// Eatches the handler.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="handler">

```



```

127     /// <para>The handler.</para>
128     /// <para></para>
129     /// </param>
130     /// <param name="restriction">
131     /// <para>The substitution.</para>
132     /// <para></para>
133     /// </param>
134     /// <returns>
135     /// <para>The link address</para>
136     /// <para></para>
137     /// </returns>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     public TLinkAddress Each(ICollection<TLinkAddress> restriction, ReadHandler<TLinkAddress>
        → handler) => SyncRoot.ExecuteReadOperation(restriction, handler, Unsync.Each);

140
141     /// <summary>
142     /// <para>
143     /// Creates the substitution.
144     /// </para>
145     /// <para></para>
146     /// </summary>
147     /// <param name="substitution">
148     /// <para>The substitution.</para>
149     /// <para></para>
150     /// </param>
151     /// <returns>
152     /// <para>The link address</para>
153     /// <para></para>
154     /// </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 =
26             ↪ Default<LinksConstants<ulong>>.Instance;
27
28         /// <summary>
29         /// <para>
30         /// Determines whether any link is any.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="links">
35         /// <para>The links.</para>
36         /// <para></para>
37         /// </param>
38         /// <param name="sequence">
39         /// <para>The sequence.</para>
40         /// <para></para>
41         /// </param>
42         /// <returns>
43         /// <para>The bool</para>
44         /// <para></para>
45         /// </returns>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static bool AnyLinkIsAny(this ILinks<ulong> links, params ulong[] sequence)
48         {
49             if (sequence == null)
50             {
51                 return false;
52             }
53             var constants = links.Constants;
54             for (var i = 0; i < sequence.Length; i++)
55             {
56                 if (sequence[i] == constants.Any)
57                 {
58                     return true;
59                 }
60             }
61             return false;
62         }
63
64         /// <summary>
65         /// <para>
66         /// Formats the structure using the specified links.
67         /// </para>
68         /// <para></para>
69         /// </summary>
70         /// <param name="links">

```

```

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

```

```

142         return sb.ToString();
143     }
144
145     /// <summary>
146     /// <para>
147     /// Appends the structure using the specified links.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <param name="links">
152     /// <para>The links.</para>
153     /// <para></para>
154     /// </param>
155     /// <param name="sb">
156     /// <para>The sb.</para>
157     /// <para></para>
158     /// </param>
159     /// <param name="visited">
160     /// <para>The visited.</para>
161     /// <para></para>
162     /// </param>
163     /// <param name="linkIndex">
164     /// <para>The link index.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="isElement">
168     /// <para>The is element.</para>
169     /// <para></para>
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
        ↳ пересечениями идентификаторов.
275 ///
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
        ↳ константой.
286 /// 3. При подтверждении транзакции (Commit) все трансформации записываются разом
        ↳ создавая задержку.
287 ///
288 /// На жёстком диске:
289 /// Минусы:
290 /// 1. Длительный отклик, на запись каждой трансформации.
291 /// 2. Лог транзакций дополнительно наполняется отменёнными транзакциями.
292 /// -> Это может решаться упаковкой/исключением дублирующих операций.
293 /// -> Также это может решаться тем, что короткие транзакции вообще
294 /// не будут записываться в случае отката.
295 /// 3. Перед тем как выполнять отмену операций транзакции нужно дождаться пока все
        ↳ операции (трансформации)
        ↳ будут записаны в лог.
296 ///
297 ///
298 /// </remarks>
299 public class Transaction : DisposableBase
300 {
301     private readonly Queue<Transition> _transitions;
302     private readonly UInt64LinksTransactionsLayer _layer;
303     /// <summary>
304     /// <para>
305     /// Gets or sets the is committed value.
306     /// </para>
307     /// <para></para>
308     /// </summary>
309     public bool IsCommitted { get; private set; }
310     /// <summary>
311     /// <para>
312     /// Gets or sets the is reverted value.
313     /// </para>
314     /// <para></para>
315     /// </summary>
316     public bool IsReverted { get; private set; }
317
318     /// <summary>

```

```

319     /// <para>
320     /// Initializes a new <see cref="Transaction"/> instance.
321     /// </para>
322     /// <para></para>
323     /// </summary>
324     /// <param name="layer">
325     /// <para>A layer.</para>
326     /// <para></para>
327     /// </param>
328     /// <exception cref="NotSupportedException">
329     /// <para>Nested transactions not supported.</para>
330     /// <para></para>
331     /// </exception>
332     [MethodImpl(MethodImplOptions.AggressiveInlining)]
333     public Transaction(UInt64LinksTransactionsLayer layer)
334     {
335         _layer = layer;
336         if (_layer._currentTransactionId != 0)
337         {
338             throw new NotSupportedException("Nested transactions not supported.");
339         }
340         IsCommitted = false;
341         IsReverted = false;
342         _transitions = new Queue<Transition>();
343         SetCurrentTransaction(layer, this);
344     }
345
346     /// <summary>
347     /// <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;

```

```

396         layer._currentTransaction = transaction;
397     }
398
399     /// <summary>
400     /// <para>
401     /// Ensures the transaction allows write operations using the specified transaction.
402     /// </para>
403     /// <para></para>
404     /// </summary>
405     /// <param name="transaction">
406     /// <para>The transaction.</para>
407     /// <para></para>
408     /// </param>
409     /// <exception cref="InvalidOperationException">
410     /// <para>Transation is committed.</para>
411     /// <para></para>
412     /// </exception>
413     /// <exception cref="InvalidOperationException">
414     /// <para>Transation is reverted.</para>
415     /// <para></para>
416     /// </exception>
417     [MethodImpl(MethodImplOptions.AggressiveInlining)]
418     public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
419     {
420         if (transaction.IsReverted)
421         {
422             throw new InvalidOperationException("Transation is reverted.");
423         }
424         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 /// <para></para>
540 /// </summary>
541 /// <param name="link">
542 /// <para>The link.</para>
543 /// <para></para>
544 /// </param>
545 /// <returns>
546 /// <para>A list of ulong</para>
547 /// <para></para>
548 /// </returns>
549 [MethodImpl(MethodImplOptions.AggressiveInlining)]
550 public IList<ulong> GetLinkValue(ulong link) => _links.GetLink(link);
551
552 /// <summary>

```

```

551     /// <para>
552     /// Creates the substitution.
553     /// </para>
554     /// <para></para>
555     /// </summary>
556     /// <param name="substitution">
557     /// <para>The substitution.</para>
558     /// <para></para>
559     /// </param>
560     /// <returns>
561     /// <para>The created link index.</para>
562     /// <para></para>
563     /// </returns>
564     [MethodImpl(MethodImplOptions.AggressiveInlining)]
565     public override ulong Create(IList<ulong> substitution, WriteHandler<ulong> handler)
566     {
567         return _links.Create(new Link<ulong>(), (before, after) =>
568         {
569             CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
570                 ↪ new Link<ulong>(before), new Link<ulong>(after)));
571             return handler(before, after);
572         });
573     }
574     /// <summary>
575     /// <para>
576     /// Updates the substitution.
577     /// </para>
578     /// <para></para>
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)]

```

```

624 private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
    ↳ _transitions;
625 [MethodImpl(MethodImplOptions.AggressiveInlining)]
626 private void CommitTransition(Transition transition)
627 {
628     if (_currentTransaction != null)
629     {
630         Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
631     }
632     var transitions = GetCurrentTransitions();
633     transitions.Enqueue(transition);
634 }
635 [MethodImpl(MethodImplOptions.AggressiveInlining)]
636 private void RevertTransition(Transition transition)
637 {
638     if (transition.After.IsNull()) // Revert Deletion with Creation
639     {
640         _links.Create();
641     }
642     else if (transition.Before.IsNull()) // Revert Creation with Deletion
643     {
644         _links.Delete(transition.After.Index);
645     }
646     else // Revert Update
647     {
648         _links.Update(new[] { transition.After.Index, transition.Before.Source,
    ↳ transition.Before.Target });
649     }
650 }
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
745 #endregion
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().Test
39             ↳ stMultipleRandomCreationsAndDeletions(100));
40         Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
41             ↳ MultipleRandomCreationsAndDeletions(100));
42         Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
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 }

```

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         links.DeleteAllUsages(root);
48     }
49     Assert.Equal(3U, links.Count());
50 }
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))

```

```

43     using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
44         ↪ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
45     {
46         memoryAdapter.TestNonexistentReferences();
47     }
48 private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
49 {
50     var link = memoryAdapter.Create();
51     memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);
52     var resultLink = _constants.Null;
53     memoryAdapter.Each(foundLink =>
54     {
55         resultLink = foundLink[_constants.IndexPart];
56         return _constants.Break;
57     }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
58     Assert.True(resultLink == link);
59     Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
60     memoryAdapter.Delete(link);
61 }
62 }
63 }

```

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     }
57 }

```

```

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
33             ↪ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
34             ↪ implementation of tree cuts out 5 bits from the address space.
35             Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
36             ↪ stMultipleRandomCreationsAndDeletions(100));
37             Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
38             ↪ MultipleRandomCreationsAndDeletions(100));
39             Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
40             ↪ tMultipleRandomCreationsAndDeletions(100));
41         }
42         private static void Using<TLink>(Action<ILinks<TLink>> action)
43         {
44             using (var dataMemory = new HeapResizableDirectMemory())
45             using (var indexMemory = new HeapResizableDirectMemory())
46             using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory))
47             {
48                 action(memory);
49             }
50         }
51         private static void UsingWithExternalReferences<TLink>(Action<ILinks<TLink>> action)
52         {
53             var contants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
54             using (var dataMemory = new HeapResizableDirectMemory())
55             using (var indexMemory = new HeapResizableDirectMemory())
56             using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory,
57             ↪ SplitMemoryLinks<TLink>.DefaultLinksSizeStep, contants))
58             {
59                 action(memory);
60             }
61         }
62     }
63 }

```

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().TestMultipleRandomCreationsAndDeletions(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, constants))
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().TestMultipleRandomCreationsAndDeletions(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 UInt64SplitMemoryLinks(dataMemory, indexMemory))
33             {
34                 action(memory);
35             }
36         }
37     }
38 }

```

```

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 UInt64SplitMemoryLinks(dataMemory, indexMemory,
43         ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, constants))
44     {
45         action(memory);
46     }
47 }
48 }

```

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             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++)
177     {
178         var random = new System.Random(N);
179         var created = 0UL;
180         var deleted = 0UL;
181         for (var i = 0; i < N; i++)
182         {
183             var linksCount = addressToUInt64Converter.Convert(links.Count());
184             var createPoint = random.NextBoolean();
185             if (linksCount >= 2 && createPoint)
186             {
187                 var linksAddressRange = new Range<ulong>(1, linksCount);
188                 TLink source = uint64ToAddressConverter.Convert(random.NextUInt64(linksA_
189                     ↪ ddressRange));
190                 TLink target = uint64ToAddressConverter.Convert(random.NextUInt64(linksA_
191                     ↪ ddressRange));
192                 ↪ //-V3086
193                 var resultLink = links.GetOrCreate(source, target);
194                 if (comparer.Compare(resultLink,
195                     ↪ uint64ToAddressConverter.Convert(linksCount)) > 0)
196                 {
197                     created++;
198                 }
199             }
200             else
201             {
202                 links.Create();
203                 created++;
204             }
205         }
206         Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
207         for (var i = 0; i < N; i++)
208         {
209             TLink link = uint64ToAddressConverter.Convert((ulong)i + 1UL);
210             if (links.Exists(link))
211             {
212                 links.Delete(link);
213                 deleted++;
214             }
215         }
216         Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
217     }
218 }
219
220 }

```

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);
34             var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
35                 ↳ markerIndex));
36             AddressToRawNumberConverter<TLink> addressToNumberConverter = new();
37             var numberAddress = addressToNumberConverter.Convert(1);
38             var numberLink = links.GetOrCreate(numberMarker, numberAddress);
39             var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
40                 ↳ true);
41             Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
42         }
43     }
44 }

```

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
29                 ↳ leRandomCreationsAndDeletions(100));
30         }
31         private static void Using(Action<ILinks<TLink>> action)
32         {

```



```

32         using (var scope = new Scope<Types<HeapResizableDirectMemory,
33             ↳ UInt32UnitedMemoryLinks>>())
34         {
35             action(scope.Use<ILinks<TLink>>());
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`, 455
`./csharp/Platform.Data.Doublets.Tests/ILinksBasicTests.cs`, 456
`./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs`, 457
`./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs`, 457
`./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs`, 458
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs`, 459
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs`, 459
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs`, 460
`./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs`, 461
`./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs`, 463
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs`, 464
`./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`, 38
`./csharp/Platform.Data.Doublets/ILinksExtensions.cs`, 38
`./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs`, 58
`./csharp/Platform.Data.Doublets/Link.cs`, 59
`./csharp/Platform.Data.Doublets/LinkExtensions.cs`, 66
`./csharp/Platform.Data.Doublets/LinksOperatorBase.cs`, 67
`./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs`, 67
`./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs`, 68
`./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs`, 69
`./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs`, 70
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 72
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs`, 79
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 85
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 89
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 93
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 97
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 101
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs`, 107
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesLinkedListMethods.cs`, 112
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 117
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs`, 121
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 124
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs`, 128
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs`, 132
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs`, 135
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs`, 153
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs`, 156
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs`, 157
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 159
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase.cs`, 165
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 171
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 175

./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 179
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs, 183
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 187
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 192
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 198
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 199
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 203
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 206
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 210
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 214
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs, 220
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 221
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase.cs, 227
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 233
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods.cs, 237
./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, 254
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 260
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 261
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 265
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 268
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 272
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 275
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 282
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 283
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 292
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 299
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 305
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 310
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 314
./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, 330
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs, 333
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs, 346
./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs, 349
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 351
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 356
./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, 369
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 373
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 377
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 382
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 383
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 391
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 396
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 402
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 407
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 411
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 414
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 420
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 423
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 427
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs, 433
./csharp/Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs, 434
./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs, 435
./csharp/Platform.Data.Doublets/Stacks/Stack.cs, 437
./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs, 438

./csharp/Platform.Data.Doublets/SynchronizedLinks.cs, 439
./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs, 442
./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 445