

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{TLinkAddress}"/>
16    /// <seealso cref="ICriterionMatcher{TLinkAddress}"/>
17    public class TargetMatcher<TLinkAddress> : LinksOperatorBase<TLinkAddress>,
18    ↪ ICriterionMatcher<TLinkAddress> where TLinkAddress : struct
19    {
20        private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
21        ↪ EqualityComparer<TLinkAddress>.Default;
22        private readonly TLinkAddress _targetToMatch;
23
24        /// <summary>
25        /// <para>
26        /// Initializes a new <see cref="TargetMatcher"/> instance.
27        /// </para>
28        /// <para></para>
29        /// </summary>
30        /// <param name="links">
31        /// <para>A links.</para>
32        /// </param>
33        /// <param name="targetToMatch">
34        /// <para>A target to match.</para>
35        /// </param>
36        [MethodImpl(MethodImplOptions.AggressiveInlining)]
37        public TargetMatcher(ILinks<TLinkAddress> links, TLinkAddress targetToMatch) :
38        ↪ base(links) => _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        /// <para></para>
52        /// </returns>
53        [MethodImpl(MethodImplOptions.AggressiveInlining)]
54        public bool IsMatched(TLinkAddress link) =>
55        ↪ _equalityComparer.Equals(_links.GetTarget(link), _targetToMatch);
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{TLinkAddress}"/>
17  public class LinksCascadeUniquenessAndUsagesResolver<TLinkAddress> :
    ↳ LinksUniquenessResolver<TLinkAddress> where TLinkAddress : struct
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<TLinkAddress> 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 TLinkAddress ResolveAddressChangeConflict(TLinkAddress
        ↳ oldLinkAddress, TLinkAddress newLinkAddress, WriteHandler<TLinkAddress>? handler)
52      {
53          var constants = _links.Constants;
54          WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
        ↳ constants.Break, handler);
55          // Use Facade (the last decorator) to ensure recursion working correctly
56          handlerState.Apply(_facade.MergeUsages(oldLinkAddress, newLinkAddress,
        ↳ handlerState.Handler));
57          handlerState.Apply(base.ResolveAddressChangeConflict(oldLinkAddress, newLinkAddress,
        ↳ handlerState.Handler));
58          return handlerState.Result;
59      }
60  }
61 }

```

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<TLinkAddress> : LinksDecoratorBase<TLinkAddress>
        ↳ where TLinkAddress : struct
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<TLinkAddress> 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 TLinkAddress Delete(IList<TLinkAddress>? restriction,
41     ↪ WriteHandler<TLinkAddress>? handler)
42     {
43         var constants = _links.Constants;
44         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
45         ↪ constants.Break, handler);
46         var linkIndex = restriction[_constants.IndexPart];
47         // Use Facade (the last decorator) to ensure recursion working correctly
48         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
49         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
50         return handlerState.Result;
51     }
52 }

```

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{TLinkAddress}"/>
17     /// <seealso cref="ILinks{TLinkAddress}"/>
18     public abstract class LinksDecoratorBase<TLinkAddress> : LinksOperatorBase<TLinkAddress>,
19     ↪ ILinks<TLinkAddress>
20     {
21         /// <summary>
22         /// <para>
23         /// The constants.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         protected readonly LinksConstants<TLinkAddress> _constants;
28
29         /// <summary>
30         /// <para>
31         /// Gets the constants value.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public LinksConstants<TLinkAddress> Constants
36         {
37             [MethodImpl(MethodImplOptions.AggressiveInlining)]
38             get => _constants;
39         }
40
41         /// <summary>
42         /// <para>
43         /// The facade.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         protected ILinks<TLinkAddress> _facade;

```

```

47
48 /// <summary>
49 /// <para>
50 /// Gets or sets the facade value.
51 /// </para>
52 /// <para></para>
53 /// </summary>
54 public ILinks<TLinkAddress> Facade
55 {
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     get => _facade;
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     set
60     {
61         _facade = value;
62         if (_links is LinksDecoratorBase<TLinkAddress> 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<TLinkAddress> 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 TLinkAddress Count(ICollection<TLinkAddress>? restriction) =>
102     ↪ _links.Count(restriction);
103
104 /// <summary>
105 /// <para>
106 /// Eaches the handler.
107 /// </para>
108 /// <para></para>
109 /// </summary>
110 /// <param name="handler">
111 /// <para>The handler.</para>
112 /// <para></para>
113 /// </param>
114 /// <param name="restriction">
115 /// <para>The restriction.</para>
116 /// <para></para>
117 /// </param>
118 /// <returns>
119 /// <para>The link</para>
120 /// <para></para>
121 /// </returns>
122 [MethodImpl(MethodImplOptions.AggressiveInlining)]
123 public virtual TLinkAddress Each(ICollection<TLinkAddress>? restriction,
124     ↪ ReadHandler<TLinkAddress>? handler) => _links.Each(restriction, handler);

```

```

123
124     /// <summary>
125     /// <para>
126     /// Creates the restriction.
127     /// </para>
128     /// <para></para>
129     /// </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 TLinkAddress Create(IList<TLinkAddress>? substitution,
140     ↪ WriteHandler<TLinkAddress>? handler) => _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 TLinkAddress Update(IList<TLinkAddress>? restriction,
162     ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler) =>
163     ↪ _links.Update(restriction, substitution, handler);
164
165     /// <summary>
166     /// <para>
167     /// Deletes the restriction.
168     /// </para>
169     /// <para></para>
170     /// </summary>
171     /// <param name="restriction">
172     /// <para>The restriction.</para>
173     /// <para></para>
174     /// </param>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     public virtual TLinkAddress Delete(IList<TLinkAddress>? restriction,
177     ↪ WriteHandler<TLinkAddress>? handler) => _links.Delete(restriction, handler);
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{TLinkAddress}"/>
16     /// <seealso cref="ILinks{TLinkAddress}"/>
17     /// <seealso cref="System.IDisposable"/>
18     public abstract class LinksDisposableDecoratorBase<TLinkAddress> :
19     ↪ LinksDecoratorBase<TLinkAddress>, ILinks<TLinkAddress>, System.IDisposable

```

```

19 {
20     /// <summary>
21     /// <para>
22     /// Represents the disposable with multiple calls allowed.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <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<TLinkAddress> links) : base(links) =>
75     ↪ Disposable = 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>

```

```

96     /// <para></para>
97     /// </param>
98     /// <param name="wasDisposed">
99     /// <para>The was disposed.</para>
100    /// <para></para>
101    /// </param>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    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{TLinkAddress}"/>
19     public class LinksInnerReferenceExistenceValidator<TLinkAddress> :
20     // ↳ LinksDecoratorBase<TLinkAddress> where TLinkAddress : struct
21     {
22         /// <summary>
23         /// <para>
24         /// Initializes a new <see cref="LinksInnerReferenceExistenceValidator"/> 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 LinksInnerReferenceExistenceValidator(ILinks<TLinkAddress> links) : base(links) {
34         // ↳ }
35
36         /// <summary>
37         /// <para>
38         /// Eaches the handler.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="handler">
43         /// <para>The handler.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="restriction">
47         /// <para>The restriction.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The link</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public override TLinkAddress Each(ICollection<TLinkAddress>? restriction,
56         // ↳ ReadHandler<TLinkAddress>? handler)
57         {
58             var links = _links;
59             links.EnsureInnerReferenceExists(restriction, nameof(restriction));
60             return links.Each(restriction, handler);
61         }
62     }
63 }

```

```

57     }
58
59     /// <summary>
60     /// <para>
61     /// Updates the restriction.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="restriction">
66     /// <para>The restriction.</para>
67     /// <para></para>
68     /// </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 TLinkAddress Update(IList<TLinkAddress>? restriction,
79     ↪  IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
80     {
81         // TODO: Possible values: null, ExistentLink or NonExistentHybrid(ExternalReference)
82         var links = _links;
83         links.EnsureInnerReferenceExists(restriction, nameof(restriction));
84         links.EnsureInnerReferenceExists(substitution, nameof(substitution));
85         return links.Update(restriction, substitution, handler);
86     }
87
88     /// <summary>
89     /// <para>
90     /// Deletes the restriction.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="restriction">
95     /// <para>The restriction.</para>
96     /// <para></para>
97     /// </param>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
100     ↪  WriteHandler<TLinkAddress>? handler)
101     {
102         var link = restriction[_constants.IndexPart];
103         var links = _links;
104         links.EnsureLinkExists(link, nameof(link));
105         return links.Delete(restriction, handler);
106     }
107 }

```

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{TLinkAddress}"/>
17     public class LinksItselfConstantToSelfReferenceResolver<TLinkAddress> :
18     ↪  LinksDecoratorBase<TLinkAddress> where TLinkAddress : struct
19     {
20         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
21         ↪  EqualityComparer<TLinkAddress>.Default;
22
23         /// <summary>
24         /// <para>
25         /// 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<TLinkAddress> 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 TLinkAddress Each(IList<TLinkAddress>? restriction,
        ↳ ReadHandler<TLinkAddress>? handler)
54     {
55         var constants = _constants;
56         var itselfConstant = constants.Itself;
57         if (!_equalityComparer.Equals(constants.Any, itselfConstant) &&
            ↳ restriction.Contains(itselfConstant))
58         {
59             // Itself constant is not supported for Each method right now, skipping execution
60             return constants.Continue;
61         }
62         return _links.Each(restriction, handler);
63     }
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 TLinkAddress Update(IList<TLinkAddress>? restriction,
        ↳ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler) =>
        ↳ _links.Update(restriction, _links.ResolveConstantAsSelfReference(_constants.Itself,
            ↳ restriction, substitution), handler);
85     }
86 }

```

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<TLinkAddress> :
18     ↪ LinksDecoratorBase<TLinkAddress> where TLinkAddress : struct
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="LinksNonExistentDependenciesCreator"/> 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 LinksNonExistentDependenciesCreator(ILinks<TLinkAddress> links) : base(links) { }
32
33         /// <summary>
34         /// <para>
35         /// Updates the restriction.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="restriction">
40         /// <para>The restriction.</para>
41         /// <para></para>
42         /// </param>
43         /// <param name="substitution">
44         /// <para>The substitution.</para>
45         /// <para></para>
46         /// </param>
47         /// <returns>
48         /// <para>The link</para>
49         /// <para></para>
50         /// </returns>
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         public override TLinkAddress Update(IList<TLinkAddress>? restriction,
53         ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
54         {
55             var constants = _constants;
56             var links = _links;
57             links.EnsureCreated(substitution[constants.SourcePart],
58             ↪ substitution[constants.TargetPart]);
59             return links.Update(restriction, substitution, handler);
60         }
61     }
62 }

```

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{TLinkAddress}"/>
17     public class LinksNullConstantToSelfReferenceResolver<TLinkAddress> :
18     ↪ LinksDecoratorBase<TLinkAddress> where TLinkAddress : struct
19     {
20         /// <summary>
21         /// <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<TLinkAddress> 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 TLinkAddress Create(ICollection<TLinkAddress>? substitution,
        ↪ WriteHandler<TLinkAddress>? 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 TLinkAddress Update(ICollection<TLinkAddress>? restriction,
        ↪ ICollection<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler) =>
        ↪ _links.Update(restriction, _links.ResolveConstantAsSelfReference(_constants.Null,
        ↪ restriction, substitution), handler);
72 }
73 }

```

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{TLinkAddress}"/>
17     public class LinksUniquenessResolver<TLinkAddress> : LinksDecoratorBase<TLinkAddress> where
        ↪ TLinkAddress : struct

```

```

18 {
19     private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
20         ↳ EqualityComparer<TLinkAddress>.Default;
21
22     /// <summary>
23     /// <para>
24     /// Initializes a new <see cref="LinksUniquenessResolver"/> instance.
25     /// </para>
26     /// </summary>
27     /// <param name="links">
28     /// <para>A links.</para>
29     /// </para>
30     /// </param>
31     [MethodImpl(MethodImplOptions.AggressiveInlining)]
32     public LinksUniquenessResolver(ILinks<TLinkAddress> links) : base(links) { }
33
34     /// <summary>
35     /// <para>
36     /// Updates the restriction.
37     /// </para>
38     /// </summary>
39     /// <param name="restriction">
40     /// <para>The restriction.</para>
41     /// </para>
42     /// </param>
43     /// <param name="substitution">
44     /// <para>The substitution.</para>
45     /// </para>
46     /// </param>
47     /// <returns>
48     /// <para>The link</para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public override TLinkAddress Update(IList<TLinkAddress>? restriction,
52         ↳ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
53     {
54         var constants = _constants;
55         var links = _links;
56         var newLinkAddress = links.SearchOrDefault(substitution[constants.SourcePart],
57             ↳ substitution[constants.TargetPart]);
58         if (_equalityComparer.Equals(newLinkAddress, default))
59         {
60             return links.Update(restriction, substitution, handler);
61         }
62         return ResolveAddressChangeConflict(restriction[constants.IndexPart],
63             ↳ newLinkAddress, handler);
64     }
65
66     /// <summary>
67     /// <para>
68     /// Resolves the address change conflict using the specified old link address.
69     /// </para>
70     /// </summary>
71     /// <param name="oldLinkAddress">
72     /// <para>The old link address.</para>
73     /// </para>
74     /// </param>
75     /// <param name="newLinkAddress">
76     /// <para>The new link address.</para>
77     /// </para>
78     /// </param>
79     /// <returns>
80     /// <para>The new link address.</para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected virtual TLinkAddress ResolveAddressChangeConflict(TLinkAddress oldLinkAddress,
84         ↳ TLinkAddress newLinkAddress, WriteHandler<TLinkAddress>? handler)
85     {
86         if (!_equalityComparer.Equals(oldLinkAddress, newLinkAddress) &&
87             ↳ _links.Exists(oldLinkAddress))
88         {
89             return _facade.Delete(oldLinkAddress, handler);
90         }
91     }
92 }

```

```

90         return _links.Constants.Continue;
91     }
92 }
93 }

```

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{TLinkAddress}"/>
17     public class LinksUniquenessValidator<TLinkAddress> : LinksDecoratorBase<TLinkAddress> where
18         ↪ TLinkAddress : struct
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="LinksUniquenessValidator"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// </param>
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public LinksUniquenessValidator(ILinks<TLinkAddress> 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 TLinkAddress Update(IList<TLinkAddress>? restriction,
52         ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? 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         }
60     }
61 }

```

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{TLinkAddress}"/>
17  public class LinksUsagesValidator<TLinkAddress> : LinksDecoratorBase<TLinkAddress> where
    ↪ TLinkAddress : struct
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<TLinkAddress> 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 TLinkAddress Update(IList<TLinkAddress>? restriction,
    ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? 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 TLinkAddress Delete(IList<TLinkAddress>? restriction,
    ↪ WriteHandler<TLinkAddress>? 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<TLinkAddress> : LinksDecoratorBase<TLinkAddress> where
        ↳ TLinkAddress : struct
8     {
9         private readonly Stream _logStream;
10        private readonly StreamWriter _logStreamWriter;
11        public LoggingDecorator(ILinks<TLinkAddress> links, Stream logStream) : base(links)
12        {
13            _logStream = logStream;
14            _logStreamWriter = new StreamWriter(_logStream);
15            _logStreamWriter.AutoFlush = true;
16        }
17
18        public override TLinkAddress Create(IList<TLinkAddress>? substitution,
        ↳ WriteHandler<TLinkAddress>? handler)
19        {
20            WriteHandlerState<TLinkAddress> handlerState = new(_constants.Continue,
        ↳ _constants.Break, handler);
21            return base.Create(substitution, (before, after) =>
22            {
23                handlerState.Handle(before, after);
24                _logStreamWriter.WriteLine($"Create. Before: {new Link<TLinkAddress>(before)}.
        ↳ After: {new Link<TLinkAddress>(after)}");
25                return _constants.Continue;
26            });
27        }
28
29        public override TLinkAddress Update(IList<TLinkAddress>? restriction,
        ↳ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
30        {
31            WriteHandlerState<TLinkAddress> handlerState = new(_constants.Continue,
        ↳ _constants.Break, handler);
32            return base.Update(restriction, substitution, (before, after) =>
33            {
34                handlerState.Handle(before, after);
35                _logStreamWriter.WriteLine($"Update. Before: {new Link<TLinkAddress>(before)}.
        ↳ After: {new Link<TLinkAddress>(after)}");
36                return _constants.Continue;
37            });
38        }
39
40        public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
        ↳ WriteHandler<TLinkAddress>? handler)
41        {
42            WriteHandlerState<TLinkAddress> handlerState = new(_constants.Continue,
        ↳ _constants.Break, handler);
43            return base.Delete(restriction, (before, after) =>
44            {
45                handlerState.Handle(before, after);
46                _logStreamWriter.WriteLine($"Delete. Before: {new Link<TLinkAddress>(before)}.
        ↳ After: {new Link<TLinkAddress>(after)}");
47                return _constants.Continue;
48            });
49        }
50    }
51 }

```

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{TLinkAddress}"/>
17     public class NonNullContentsLinkDeletionResolver<TLinkAddress> :
        ↳ LinksDecoratorBase<TLinkAddress> where TLinkAddress : struct
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<TLinkAddress> 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 TLinkAddress Delete(IList<TLinkAddress>? restriction,
44     ↪ WriteHandler<TLinkAddress>? handler)
45     {
46         var linkIndex = restriction[_constants.IndexPart];
47         var constants = _links.Constants;
48         WriteHandlerState<TLinkAddress> handlerResult = new(constants.Continue,
49         ↪ constants.Break, handler);
50         handlerResult.Apply(_links.EnforceResetValues(linkIndex, handlerResult.Handler));
51         handlerResult.Apply(_links.Delete(restriction, handlerResult.Handler));
52         return handlerResult.Result;
53     }
54 }

```

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 TLinkAddress = 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{TLinkAddress}"/>
18     public class UInt32Links : LinksDisposableDecoratorBase<TLinkAddress>
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<TLinkAddress> 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 TLinkAddress Create(ICollection<TLinkAddress>? substitution,
    ↪ WriteHandler<TLinkAddress>? handler) => _links.CreatePoint(handler);
49
50    /// <summary>
51    /// <para>
52    /// Updates the substitution.
53    /// </para>
54    /// <para></para>
55    /// </summary>
56    /// <param name="restriction">
57    /// <para>The substitution.</para>
58    /// <para></para>
59    /// </param>
60    /// <param name="substitution">
61    /// <para>The substitution.</para>
62    /// <para></para>
63    /// </param>
64    /// <returns>
65    /// <para>The link</para>
66    /// <para></para>
67    /// </returns>
68    [MethodImpl(MethodImplOptions.AggressiveInlining)]
69    public override TLinkAddress Update(ICollection<TLinkAddress>? restriction,
    ↪ ICollection<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
70    {
71        var constants = _constants;
72        var indexPartConstant = constants.IndexPart;
73        var sourcePartConstant = constants.SourcePart;
74        var targetPartConstant = constants.TargetPart;
75        var nullConstant = constants.Null;
76        var itselfConstant = constants.Itself;
77        var existedLink = nullConstant;
78        var updatedLink = restriction[indexPartConstant];
79        var newSource = substitution[sourcePartConstant];
80        var newTarget = substitution[targetPartConstant];
81        var links = _links;
82        if (newSource != itselfConstant && newTarget != itselfConstant)
83        {
84            existedLink = links.SearchOrDefault(newSource, newTarget);
85        }
86        if (existedLink == nullConstant)
87        {
88            var before = links.GetLink(updatedLink);
89            if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
    ↪ newTarget)
90            {
91                var source = newSource == itselfConstant ? updatedLink : newSource;
92                var target = newTarget == itselfConstant ? updatedLink : newTarget;
93                return links.Update(new Link<TLinkAddress>(updatedLink, source, target),
    ↪ handler);
94            }
95            return _links.Constants.Continue;
96        }
97        else
98        {
99            return _facade.MergeAndDelete(updatedLink, existedLink, handler);
100        }
101    }
102
103    /// <summary>
104    /// <para>
105    /// Deletes the substitution.
106    /// </para>
107    /// <para></para>
108    /// </summary>
109    /// <param name="restriction">
110    /// <para>The substitution.</para>
111    /// <para></para>
112    /// </param>
113    [MethodImpl(MethodImplOptions.AggressiveInlining)]
114    public override TLinkAddress Delete(ICollection<TLinkAddress>? restriction,
    ↪ WriteHandler<TLinkAddress>? handler)

```

```

115     {
116         var linkIndex = restriction[_constants.IndexPart];
117         var constants = _links.Constants;
118         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
119             ↪ constants.Break, handler);
119         handlerState.Apply( _links.EnforceResetValues(linkIndex, handlerState.Handler));
120         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
121         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
122         return handlerState.Result;
123     }
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 TLinkAddress = 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>
15     /// <para>Представляет комбинированный декоратор, реализующий основную логику по
16     ↪ взаимодействию с хранилищем связей, для связей с адресами представленными в виде <see
17     ↪ cref="System.UInt64"></para>
18     /// </summary>
19     /// <remarks>
20     /// Возможные оптимизации:
21     /// Объединение в одном поле Source и Target с уменьшением до 32 бит.
22     /// + меньше объём БД
23     /// - меньше производительность
24     /// - больше ограничение на количество связей в БД)
25     /// Ленивое хранение размеров поддеревьев (расчитываемое по мере использования БД)
26     /// + меньше объём БД
27     /// - больше сложность
28     ///
29     /// Текущее теоретическое ограничение на индекс связи, из-за использования 5 бит в размере
30     ↪ поддеревьев для AVL баланса и флагов нитей: 2 в степени(64 минус 5 равно 59 ) равно 576
31     ↪ 460 752 303 423 488
32     /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами
33     ↪ (битовыми строками) - вариант матрицы (выстраиваемой лениво).
34     ///
35     /// Решить отключать ли проверки при компиляции под Release. Т.е. исключения будут
36     ↪ выбрасываться только при #if DEBUG
37     /// </remarks>
38     public class UInt64Links : LinksDisposableDecoratorBase<TLinkAddress>
39     {
40         /// <summary>
41         /// <para>
42         /// Initializes a new <see cref="UInt64Links"> instance.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="links">
47         /// <para>A links.</para>
48         /// <para></para>
49         /// </param>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public UInt64Links(ILinks<TLinkAddress> links) : base(links) { }
52
53         /// <summary>
54         /// <para>
55         /// Creates the substitution.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         /// <param name="substitution">
60         /// <para>The substitution.</para>
61         /// <para></para>
62         /// </param>
63         /// <returns>
64         /// <para>The TLinkAddress</para>

```

```

57     /// <para></para>
58     /// </returns>
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     public override TLinkAddress Create(ICollection<TLinkAddress>? substitution,
61     ↪ WriteHandler<TLinkAddress>? handler) => _links.CreatePoint(handler);
62
63     /// <summary>
64     /// <para>
65     /// Updates the substitution.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="restriction">
70     /// <para>The substitution.</para>
71     /// </param>
72     /// <param name="substitution">
73     /// <para>The substitution.</para>
74     /// <para></para>
75     /// </param>
76     /// <returns>
77     /// <para>The TLinkAddress</para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public override TLinkAddress Update(ICollection<TLinkAddress>? restriction,
81     ↪ ICollection<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? 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<TLinkAddress>(updatedLink, source, target),
107                 ↪ handler);
108             }
109             return _links.Constants.Continue;
110         }
111         else
112         {
113             return _facade.MergeAndDelete(updatedLink, existedLink, handler);
114         }
115     }
116
117     /// <summary>
118     /// <para>
119     /// Deletes the substitution.
120     /// </para>
121     /// <para></para>
122     /// </summary>
123     /// <param name="restriction">
124     /// <para>The substitution.</para>
125     /// </param>
126     [MethodImpl(MethodImplOptions.AggressiveInlining)]
127     public override TLinkAddress Delete(ICollection<TLinkAddress>? restriction,
128     ↪ WriteHandler<TLinkAddress>? handler)
129     {
130         var linkIndex = restriction[_constants.IndexPart];
131         var constants = _links.Constants;

```

```

130         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
            ↪ constants.Break, handler);
131         handlerState.Apply(_links.EnforceResetValues(linkIndex, handlerState.Handler));
132         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
133         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
134         return handlerState.Result;
135     }
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.)
21     /// </remarks>
22     internal class UniLinks<TLinkAddress> : LinksDecoratorBase<TLinkAddress>,
23     ↪ IUniLinks<TLinkAddress> where TLinkAddress : struct
24     {
25         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
26         ↪ EqualityComparer<TLinkAddress>.Default;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="UniLinks"/> instance.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="links">
35         /// <para>A links.</para>
36         /// <para></para>
37         /// </param>
38         public UniLinks(ILinks<TLinkAddress> links) : base(links) { }
39         private struct Transition
40         {
41             /// <summary>
42             /// <para>
43             /// The before.
44             /// </para>
45             /// <para></para>
46             /// </summary>
47             public IList<TLinkAddress>? Before;
48             /// <summary>
49             /// <para>
50             /// The after.
51             /// </para>
52             /// <para></para>
53             /// </summary>
54             public IList<TLinkAddress>? After;
55
56             /// <summary>
57             /// <para>
58             /// Initializes a new <see cref="Transition"/> instance.
59             /// </para>
60             /// <para></para>
61             /// </summary>
62             /// <param name="before">
63             /// <para>A before.</para>
64             /// <para></para>
65             /// </param>
66             /// <param name="after">
67             /// <para>A after.</para>
68             /// <para></para>
69             /// </param>

```

```

64     /// </param>
65     public Transition(IList<TLinkAddress>? before, IList<TLinkAddress>? after)
66     {
67         Before = before;
68         After = after;
69     }
70 }
71
72 //public static readonly TLinkAddress NullConstant =
73     ↪ Use<LinksConstants<TLinkAddress>>.Single.Null;
74 //public static readonly IReadOnlyList<TLinkAddress> NullLink = new
75     ↪ ReadOnlyCollection<TLinkAddress>(new List<TLinkAddress> { NullConstant,
76     ↪ NullConstant, NullConstant });
77
78 // TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
79     ↪ (Links-Expression)
80 /// <summary>
81 /// <para>
82 /// Triggers the restriction.
83 /// </para>
84 /// <para></para>
85 /// </summary>
86 /// <param name="restriction">
87 /// <para>The restriction.</para>
88 /// <para></para>
89 /// </param>
90 /// <param name="matchedHandler">
91 /// <para>The matched handler.</para>
92 /// <para></para>
93 /// </param>
94 /// <param name="substitution">
95 /// <para>The substitution.</para>
96 /// <para></para>
97 /// </param>
98 /// <param name="substitutedHandler">
99 /// <para>The substituted handler.</para>
100 /// <para></para>
101 /// </param>
102 /// <returns>
103 /// <para>The link</para>
104 /// <para></para>
105 /// </returns>
106 public TLinkAddress Trigger(IList<TLinkAddress>? restriction,
107     ↪ WriteHandler<TLinkAddress>? matchedHandler, IList<TLinkAddress>? substitution,
108     ↪ WriteHandler<TLinkAddress>? substitutedHandler)
109 {
110     ///List<Transition> transitions = null;
111     ///if (!restriction.IsNullOrEmpty())
112     ///{
113     ///    // Есть причина делать проход (чтение)
114     ///    if (matchedHandler != null)
115     ///    {
116     ///        if (!substitution.IsNullOrEmpty())
117     ///        {
118     ///            // restriction => { 0, 0, 0 } | { 0 } // Create
119     ///            // substitution => { itself, 0, 0 } | { itself, itself, itself } //
120     ///            ↪ Create / Update
121     ///            // substitution => { 0, 0, 0 } | { 0 } // Delete
122     ///            transitions = new List<Transition>();
123     ///            if (Equals(substitution[Constants.IndexPart], Constants.Null))
124     ///            {
125     ///                // If index is Null, that means we always ignore every other
126     ///                ↪ value (they are also Null by definition)
127     ///                var matchDecision = matchedHandler(, NullLink);
128     ///                if (Equals(matchDecision, Constants.Break))
129     ///                    return false;
130     ///                if (!Equals(matchDecision, Constants.Skip))
131     ///                    transitions.Add(new Transition(matchedLink, newValue));
132     ///            }
133     ///            else
134     ///            {
135     ///                Func<T, bool> handler;
136     ///                handler = link =>
137     ///                {
138     ///                    var matchedLink = Memory.GetLinkValue(link);
139     ///                    var newValue = Memory.GetLinkValue(link);
140     ///                    newValue[Constants.IndexPart] = Constants.Itself;
141     ///                }
142     ///            }
143     ///        }
144     ///    }
145     ///}

```

```

133     newLink[Constants.SourcePart] =
134     ↪ Equals(substitution[Constants.SourcePart], Constants.Itself) ?
135     ↪ matchedLink[Constants.IndexPart] : substitution[Constants.SourcePart];
136     newLink[Constants.TargetPart] =
137     ↪ Equals(substitution[Constants.TargetPart], Constants.Itself) ?
138     ↪ matchedLink[Constants.IndexPart] : substitution[Constants.TargetPart];
139     var matchDecision = matchedHandler(matchedLink, newLink);
140     if (Equals(matchDecision, Constants.Break))
141         return false;
142     if (!Equals(matchDecision, Constants.Skip))
143         transitions.Add(new Transition(matchedLink, newLink));
144     return true;
145 };
146 if (!Memory.Each(handler, restriction))
147     return Constants.Break;
148 }
149 else
150 {
151     Func<T, bool> handler = link =>
152     {
153         var matchedLink = Memory.GetLinkValue(link);
154         var matchDecision = matchedHandler(matchedLink, matchedLink);
155         return !Equals(matchDecision, Constants.Break);
156     };
157     if (!Memory.Each(handler, restriction))
158         return Constants.Break;
159 }
160 else
161 {
162     if (substitution != null)
163     {
164         transitions = new List<ILink<T>>();
165         Func<T, bool> handler = link =>
166         {
167             var matchedLink = Memory.GetLinkValue(link);
168             transitions.Add(matchedLink);
169             return true;
170         };
171         if (!Memory.Each(handler, restriction))
172             return Constants.Break;
173     }
174     else
175     {
176         return Constants.Continue;
177     }
178 }
179 }
180 }
181 if (substitution != null)
182 {
183     // Есть причина делать замену (запись)
184     if (substitutedHandler != null)
185     {
186     }
187     else
188     {
189     }
190 }
191 }
192 return Constants.Continue;
193
194 //if (restriction.IsNullOrEmpty()) // Create
195 //{
196 //    substitution[Constants.IndexPart] = Memory.AllocateLink();
197 //    Memory.SetLinkValue(substitution);
198 //}
199 //else if (restriction.IsNullOrEmpty()) // Delete
200 //{
201 //    Memory.FreeLink(restriction[Constants.IndexPart]);
202 //}
203 //else if (restriction.EqualTo(substitution)) // Read or ("repeat" the state) // Each
204 //{
205 //    // No need to collect links to list
206 //    // Skip == Continue
207 //    // No need to check substitutedHandler
208 //    if (!Memory.Each(link => !Equals(matchedHandler(Memory.GetLinkValue(link)),
209 //    ↪ Constants.Break), restriction))

```

```

205         //         return Constants.Break;
206     //}
207     //else // Update
208     //{
209         //     //List<IList<T>> matchedLinks = null;
210         //     if (matchedHandler != null)
211         //     {
212             //         matchedLinks = new List<IList<T>>();
213             //         Func<T, bool> handler = link =>
214             //         {
215                 //             var matchedLink = Memory.GetLinkValue(link);
216                 //             var matchDecision = matchedHandler(matchedLink);
217                 //             if (Equals(matchDecision, Constants.Break))
218                     //                 return false;
219                 //             if (!Equals(matchDecision, Constants.Skip))
220                     //                 matchedLinks.Add(matchedLink);
221                 //             return true;
222             //         };
223             //         if (!Memory.Each(handler, restriction))
224                 //             return Constants.Break;
225             //     }
226             //     if (!matchedLinks.IsNullOrEmpty())
227             //     {
228                 //         var totalMatchedLinks = matchedLinks.Count;
229                 //         for (var i = 0; i < totalMatchedLinks; i++)
230                 //         {
231                     //             var matchedLink = matchedLinks[i];
232                     //             if (substitutedHandler != null)
233                     //             {
234                         //                 var newValue = new List<T>(); // TODO: Prepare value to update here
235                         //                 // TODO: Decide is it actually needed to use Before and After
236                         ↪ substitution handling.
237                         //                 var substitutedDecision = substitutedHandler(matchedLink,
238                         ↪ newValue);
239                         //                 if (Equals(substitutedDecision, Constants.Break))
240                             //                     return Constants.Break;
241                         //                 if (Equals(substitutedDecision, Constants.Continue))
242                             //                 {
243                                 //                     // Actual update here
244                                 //                     Memory.SetLinkValue(newValue);
245                             //                 }
246                         //                 if (Equals(substitutedDecision, Constants.Skip))
247                             //                 {
248                                 //                     // Cancel the update. TODO: decide use separate Cancel
249                                 ↪ constant or Skip is enough?
250                                 //                 }
251                             //             }
252                         //         }
253                     //     }
254             // }
255         // }
256     //}
257     return _constants.Continue;
258 }
259
260 /// <summary>
261 /// <para>
262 /// Triggers the pattern or condition.
263 /// </para>
264 /// <para></para>
265 /// </summary>
266 /// <param name="patternOrCondition">
267 /// <para>The pattern or condition.</para>
268 /// <para></para>
269 /// </param>
270 /// <param name="matchHandler">
271 /// <para>The match handler.</para>
272 /// <para></para>
273 /// </param>
274 /// <param name="substitution">
275 /// <para>The substitution.</para>
276 /// <para></para>
277 /// </param>
278 /// <param name="substitutionHandler">
279 /// <para>The substitution handler.</para>
280 /// <para></para>
281 /// </param>
282 /// <exception cref="NotImplementedException">
283 /// <para></para>
284 /// <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 TLinkAddress Trigger(IList<TLinkAddress>? patternOrCondition,
        ↳ ReadHandler<TLinkAddress>? matchHandler, IList<TLinkAddress>? substitution,
        ↳ WriteHandler<TLinkAddress>? substitutionHandler)
302     {
303         var constants = _constants;
304         if (patternOrCondition.IsNullOrEmpty() && substitution.IsNullOrEmpty())
305         {
306             return constants.Continue;
307         }
308         else if (patternOrCondition.EqualTo(substitution)) // Should be Each here TODO:
        ↳ Check if it is a correct condition
309         {
310             // Or it only applies to trigger without matchHandler.
311             throw new NotImplementedException();
312         }
313         else if (!substitution.IsNullOrEmpty()) // Creation
314         {
315             var before = Array.Empty<TLinkAddress>();
316             // Что должно означать False здесь? Остановиться (перестать идти) или пропустить
        ↳ (пройти мимо) или пустить (взять)?
317             if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
        ↳ constants.Break))
318             {
319                 return constants.Break;
320             }
321             var after = (IList<TLinkAddress>?)substitution.ToArray();
322             if (_equalityComparer.Equals(after[0], default))
323             {
324                 var newLink = _links.Create();
325                 after[0] = newLink;
326             }
327             if (substitution.Count == 1)
328             {
329                 after = _links.GetLink(substitution[0]);
330             }
331             else if (substitution.Count == 3)
332             {
333                 //Links.Create(after);
334             }
335             else
336             {
337                 throw new NotSupportedException();
338             }
339             return matchHandler != null ? substitutionHandler(before, after) :
        ↳ constants.Continue;
340         }
341         else if (!patternOrCondition.IsNullOrEmpty()) // Deletion
342         {
343             if (patternOrCondition.Count == 1)
344             {
345                 var linkToDelete = patternOrCondition[0];
346                 var before = _links.GetLink(linkToDelete);
347                 if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
        ↳ constants.Break))
348                 {
349                     return constants.Break;
350                 }
351             }
352         }

```



```

var after = Array.Empty<TLinkAddress>();
_links.Update(linkToDelete, constants.Null, constants.Null);
_links.Delete(linkToDelete);
return matchHandler != null ? substitutionHandler(before, after) :
    ↪ constants.Continue;
}
else
{
    throw new NotSupportedException();
}
}
else // Replace / Update
{
    if (patternOrCondition.Count == 1) //-V3125
    {
        var linkToUpdate = patternOrCondition[0];
        var before = _links.GetLink(linkToUpdate);
        if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
            ↪ constants.Break))
        {
            return constants.Break;
        }
        var after = (IList<TLinkAddress>?)substitution.ToArray(); //-V3125
        if (_equalityComparer.Equals(after[0], default))
        {
            after[0] = linkToUpdate;
        }
        if (substitution.Count == 1)
        {
            if (!_equalityComparer.Equals(substitution[0], linkToUpdate))
            {
                after = _links.GetLink(substitution[0]);
                _links.Update(linkToUpdate, constants.Null, constants.Null);
                _links.Delete(linkToUpdate);
            }
        }
        else if (substitution.Count == 3)
        {
            //Links.Update(after);
        }
        else
        {
            throw new NotSupportedException();
        }
        return matchHandler != null ? substitutionHandler(before, after) :
            ↪ constants.Continue;
    }
    else
    {
        throw new NotSupportedException();
    }
}
}

/// <remarks>
/// IList[IList[IList[T]]]
/// | | |
/// | | ----- |
/// | | link |
/// | ----- |
/// | change |
/// |-----|
/// changes
/// </remarks>
public IList<IList<IList<TLinkAddress>?>> Trigger(IList<TLinkAddress>? condition,
    ↪ IList<TLinkAddress>? substitution)
{
    var changes = new List<IList<IList<TLinkAddress>?>>();
    var @continue = _constants.Continue;
    Trigger(condition, AlwaysContinue, substitution, (before, after) =>
    {
        var change = new[] { before, after };
        changes.Add(change);
        return @continue;
    });
    return changes;
}

```

```

424         private TLinkAddress AlwaysContinue(IList<TLinkAddress>? 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         /// <summary>
33         /// <para>.</para>
34         /// <para>.</para>
35         /// </summary>
36         /// <typeparam name="T">
37         /// <para>.</para>
38         /// <para>.</para>
39         /// </typeparam>
40         public readonly T Target;
41
42         /// <summary>
43         /// <para>.</para>
44         /// <para>.</para>
45         /// </summary>
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         /// <summary>
66         /// <para>.</para>
67         /// <para>.</para>
68         /// </summary>
69         /// <returns>
70         /// <para>.</para>
71         /// <para>.</para>
72         /// </returns>

```

```

73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 public override string ToString() => $"{Source}->{Target}";
75
76 /// <summary>
77 /// <para>.</para>
78 /// <para>.</para>
79 /// </summary>
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 /// <summary>
97 /// <para>.</para>
98 /// <para>.</para>
99 /// </summary>
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>
110 /// <para>.</para>
111 /// </returns>
112 [MethodImpl(MethodImplOptions.AggressiveInlining)]
113 public override bool Equals(object obj) => obj is Doublet<T> doublet ?
114     ↳ base.Equals(doublet) : false;
115
116 /// <summary>
117 /// <para>.</para>
118 /// <para>.</para>
119 /// </summary>
120 /// <returns>
121 /// <para>.</para>
122 /// <para>.</para>
123 /// </returns>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 public override int GetHashCode() => (Source, Target).GetHashCode();
126
127 /// <summary>
128 /// <para>.</para>
129 /// <para>.</para>
130 /// </summary>
131 /// <param name="left">
132 /// <para>.</para>
133 /// <para>.</para>
134 /// </param>
135 /// <param name="right">
136 /// <para>.</para>
137 /// <para>.</para>
138 /// </param>
139 /// <returns>
140 /// <para>.</para>
141 /// <para>.</para>
142 /// </returns>
143 [MethodImpl(MethodImplOptions.AggressiveInlining)]
144 public static bool operator ==(Doublet<T> left, Doublet<T> right) => left.Equals(right);
145
146 /// <summary>
147 /// <para>.</para>
148 /// <para>.</para>
149 /// </summary>
150 /// <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 TLinkAddress = System.UInt32;
12
13     public class UInt32UnitedMemoryLinks : DisposableBase, ILinks<TLinkAddress>
14     {
15         public LinksConstants<TLinkAddress> 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<TLinkAddress>(enableExternalReferencesSupport:
27                     ↪ true);
28             }
29
30             public TLinkAddress Count(IList<TLinkAddress>? 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                     return Methods.UInt32UnitedMemoryLinks_Count(_ptr, array,
40                         ↪ (nuint)(restriction?.Count ?? 0));
41                 }
42             }
43
44             public TLinkAddress Each(IList<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
45                 ↪ handler)
46             {
47                 unsafe
48                 {
49                     Methods.EachCallback_UInt32 callback = (link) => handler?.Invoke(new
50                         ↪ Link<TLinkAddress>(link.Index, link.Source, link.Target)) ??
51                         ↪ Constants.Continue;
52                     var array = stackalloc uint[restriction.Count];
53                     for (var i = 0; i < restriction.Count; i++)
54                     {
55                         array[i] = restriction[i];
56                     }
57                     return Methods.UInt32UnitedMemoryLinks_Each(_ptr, array,
58                         ↪ (nuint)(restriction?.Count ?? 0), callback);
59                 }
60             }
61
62             public TLinkAddress Create(IList<TLinkAddress>? substitution,
63                 ↪ WriteHandler<TLinkAddress>? handler)
64             {
65                 unsafe
66                 {
67                     Methods.CreateCallback_UInt32 callback = (before, after) => handler?.Invoke(new
68                         ↪ Link<TLinkAddress>(before.Index, before.Source, before.Target), new
69                         ↪ Link<TLinkAddress>(after.Index, after.Source, after.Target)) ??
70                         ↪ Constants.Continue;
71                     fixed (uint* substitutionPtr = (uint[])substitution)
72                     {
73                         return Methods.UInt32UnitedMemoryLinks_Create(_ptr, substitutionPtr,
74                             ↪ (nuint)(substitution?.Count ?? 0), callback);
75                     }
76                 }
77             }
78         }
79     }
80 }

```

```

68
69 public TLinkAddress Update(ICollection<TLinkAddress>? restriction, ICollection<TLinkAddress>?
    ↳ substitution, WriteHandler<TLinkAddress>? 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 < substitution.Count; i++)
80         {
81             substitutionArray[i] = substitution[i];
82         }
83         Methods.UpdateCallback_UInt32 callback = (before, after) => handler?.Invoke(new
            ↳ Link<TLinkAddress>(before.Index, before.Source, before.Target), new
            ↳ Link<TLinkAddress>(after.Index, after.Source, after.Target)) ??
            ↳ Constants.Continue;
84         return Methods.UInt32UnitedMemoryLinks_Update(_ptr, restrictionArray,
            ↳ (uint)(restriction?.Count ?? 0), substitutionArray,
            ↳ (uint)(substitution?.Count ?? 0), callback);
85     }
86 }
87
88 public TLinkAddress Delete(ICollection<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
    ↳ 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<TLinkAddress>(before.Index, before.Source, before.Target), new
            ↳ Link<TLinkAddress>(after.Index, after.Source, after.Target)) ??
            ↳ Constants.Continue;
98         return Methods.UInt32UnitedMemoryLinks_Delete(_ptr, restrictionArray,
            ↳ (uint)(restriction?.Count ?? 0), callback);
99     }
100 }
101
102 protected override void Dispose(bool manual, bool wasDisposed)
103 {
104     unsafe
105     {
106         if (wasDisposed || _ptr == null)
107         {
108             return;
109         }
110         Methods.UInt32UnitedMemoryLinks_Drop(_ptr);
111     }
112 }
113 }
114 }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.InteropServices;
5 using Platform.Converters;
6 using Platform.Delegates;
7 using Platform.Disposables;
8
9 namespace Platform.Data.Doublets.FFI
10 {
11     struct FfiLink_UInt8
12     {
13         public Byte Index;
14         public Byte Source;
15         public Byte Target;
16     }
17
18     struct FfiLink_UInt16
19     {

```

```

20     public UInt16 Index;
21     public UInt16 Source;
22     public UInt16 Target;
23 }
24
25 struct FfiLink_UInt32
26 {
27     public UInt32 Index;
28     public UInt32 Source;
29     public UInt32 Target;
30 }
31
32 struct FfiLink_UInt64
33 {
34     public UInt64 Index;
35     public UInt64 Source;
36     public UInt64 Target;
37 }
38
39 unsafe static class Methods
40 {
41     private const string DllName = "Platform.Doublets";
42
43     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
44     public delegate Byte EachCallback_UInt8(FfiLink_UInt8 link);
45
46     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
47     public delegate UInt16 EachCallback_UInt16(FfiLink_UInt16 link);
48
49     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
50     public delegate UInt32 EachCallback_UInt32(FfiLink_UInt32 link);
51
52     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
53     public delegate UInt64 EachCallback_UInt64(FfiLink_UInt64 link);
54
55     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
56     public delegate Byte CreateCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
57
58     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
59     public delegate UInt16 CreateCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
60         ↪ after);
61
62     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
63     public delegate UInt32 CreateCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
64         ↪ after);
65
66     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
67     public delegate UInt64 CreateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
68         ↪ after);
69
70     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
71     public delegate Byte UpdateCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
72
73     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
74     public delegate UInt16 UpdateCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
75         ↪ after);
76
77     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
78     public delegate UInt32 UpdateCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
79         ↪ after);
80
81     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
82     public delegate UInt64 UpdateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
83         ↪ after);
84
85     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
86     public delegate Byte DeleteCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
87
88     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
89     public delegate UInt16 DeleteCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
90         ↪ after);
91
92     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
93     public delegate UInt32 DeleteCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
94         ↪ after);
95
96     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
97     public delegate UInt64 DeleteCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
98         ↪ after);
99
100 }

```

```

91 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
92 public static extern void* ByteUnitedMemoryLinks_New(string path);
93
94 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
95 public static extern void* UInt16UnitedMemoryLinks_New(string path);
96
97 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
98 public static extern void* UInt32UnitedMemoryLinks_New(string path);
99
100 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
101 public static extern void* UInt64UnitedMemoryLinks_New(string path);
102
103 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
104 public static extern void ByteUnitedMemoryLinks_Drop(void* self);
105
106 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
107 public static extern void UInt16UnitedMemoryLinks_Drop(void* self);
108
109 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
110 public static extern void UInt32UnitedMemoryLinks_Drop(void* self);
111
112 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
113 public static extern void UInt64UnitedMemoryLinks_Drop(void* self);
114
115 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
116 public static extern byte ByteUnitedMemoryLinks_Create(void* self, byte* substitution,
117     ↳ nuint substitutionLength, CreateCallback UInt8 callback);
118
119 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
120 public static extern ushort UInt16UnitedMemoryLinks_Create(void* self, ushort*
121     ↳ substitution, nuint substitutionLength, CreateCallback UInt16 callback);
122
123 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
124 public static extern uint UInt32UnitedMemoryLinks_Create(void* self, uint* substitution,
125     ↳ nuint substitutionLength, CreateCallback UInt32 callback);
126
127 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
128 public static extern byte ByteUnitedMemoryLinks_Count(void* self, byte* restriction,
129     ↳ nuint len);
130
131 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
132 public static extern ushort UInt16UnitedMemoryLinks_Count(void* self, ushort*
133     ↳ restriction, nuint len);
134
135 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
136 public static extern uint UInt32UnitedMemoryLinks_Count(void* self, uint* restriction,
137     ↳ nuint len);
138
139 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
140 public static extern ulong UInt64UnitedMemoryLinks_Count(void* self, ulong* restriction,
141     ↳ nuint len);
142
143 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
144 public static extern byte ByteUnitedMemoryLinks_Each(void* self, byte* restriction,
145     ↳ nuint len, EachCallback UInt8 callback);
146
147 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
148 public static extern ushort UInt16UnitedMemoryLinks_Each(void* self, ushort*
149     ↳ restriction, nuint len, EachCallback UInt16 callback);
150
151 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
152 public static extern uint UInt32UnitedMemoryLinks_Each(void* self, uint* restriction,
153     ↳ nuint len, EachCallback UInt32 callback);
154
155 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
156 public static extern ulong UInt64UnitedMemoryLinks_Each(void* self, ulong* restriction,
157     ↳ nuint len, EachCallback UInt64 callback);
158
159 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
160 public static extern byte ByteUnitedMemoryLinks_Update(void* self, byte* restriction,
161     ↳ nuint restrictionLength, byte* substitution, nuint substitutionLength,
162     ↳ UpdateCallback UInt8 callback);
163
164 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]

```



```

public static extern ushort UInt16UnitedMemoryLinks_Update(void* self, ushort*
↪ restriction, nuint restrictionLength, ushort* substitution, nuint
↪ substitutionLength, UpdateCallback_UInt16 callback);

[DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
public static extern uint UInt32UnitedMemoryLinks_Update(void* self, uint* restriction,
↪ nuint restrictionLength, uint* substitution, nuint substitutionLength,
↪ UpdateCallback_UInt32 callback);

[DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
public static extern ulong UInt64UnitedMemoryLinks_Update(void* self, ulong*
↪ restriction, nuint restrictionLength, ulong* substitution, nuint
↪ substitutionLength, UpdateCallback_UInt64 callback);

[DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
public static extern byte ByteUnitedMemoryLinks_Delete(void* self, byte* restriction,
↪ nuint restrictionLength, DeleteCallback_UInt8 callback);

[DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
public static extern ushort UInt16UnitedMemoryLinks_Delete(void* self, ushort*
↪ restriction, nuint len, DeleteCallback_UInt16 callback);

[DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
public static extern uint UInt32UnitedMemoryLinks_Delete(void* self, uint* restriction,
↪ nuint len, DeleteCallback_UInt32 callback);

[DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
public static extern ulong UInt64UnitedMemoryLinks_Delete(void* self, ulong*
↪ restriction, nuint len, DeleteCallback_UInt64 callback);

public class UnitedMemoryLinks<TLinkAddress> : DisposableBase, ILinks<TLinkAddress> where
TLinkAddress : struct

private static readonly UncheckedConverter<byte, TLinkAddress> from_u8 =
↪ UncheckedConverter<byte, TLinkAddress>.Default;
private static readonly UncheckedConverter<ushort, TLinkAddress> from_u16 =
↪ UncheckedConverter<ushort, TLinkAddress>.Default;
private static readonly UncheckedConverter<uint, TLinkAddress> from_u32 =
↪ UncheckedConverter<uint, TLinkAddress>.Default;
private static readonly UncheckedConverter<ulong, TLinkAddress> from_u64 =
↪ UncheckedConverter<ulong, TLinkAddress>.Default;
private static readonly UncheckedConverter<TLinkAddress, ulong> from_t =
↪ UncheckedConverter<TLinkAddress, ulong>.Default;

public LinksConstants<TLinkAddress> Constants { get; }

private readonly unsafe void* _ptr;

public UnitedMemoryLinks(string path)
{
    TLinkAddress t = default;
    unsafe
    {
        _ptr = t switch
        {
            byte => Methods.ByteUnitedMemoryLinks_New(path),
            ushort => Methods.UInt16UnitedMemoryLinks_New(path),
            uint => Methods.UInt32UnitedMemoryLinks_New(path),
            ulong => Methods.UInt64UnitedMemoryLinks_New(path),
            _ => throw new NotImplementedException()
        };

        // TODO: Update api
        Constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
↪ true);
    }
}

public TLinkAddress Count(ICollection<TLinkAddress>? restriction)
{
    var restrictionLength = restriction?.Count ?? 0;
    unsafe
    {
        TLinkAddress t = default;
        switch (t)
        {
            case byte:
            {

```

```

217         var restrictionArray = stackalloc byte[restrictionLength];
218         var byteRestrictionArray = (IList<byte>)restriction;
219         for (var i = 0; i < restrictionLength; i++)
220         {
221             restrictionArray[i] = byteRestrictionArray[i];
222         }
223         return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Count(_ptr,
224                               ↪ restrictionArray, (nuint)restrictionLength));
225     case ushort:
226     {
227         var restrictionArray = stackalloc ushort[restrictionLength];
228         var ushortRestrictionArray = (IList<ushort>)restriction;
229         for (var i = 0; i < restrictionLength; i++)
230         {
231             restrictionArray[i] = ushortRestrictionArray[i];
232         }
233         return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Count(_ptr,
234                               ↪ restrictionArray, (nuint)restrictionLength));
235     }
236     case uint:
237     {
238         var restrictionArray = stackalloc uint[restrictionLength];
239         var uintRestrictionArray = (IList<uint>)restriction;
240         for (var i = 0; i < restrictionLength; i++)
241         {
242             restrictionArray[i] = uintRestrictionArray[i];
243         }
244         return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Count(_ptr,
245                               ↪ restrictionArray, (nuint)restrictionLength));
246     }
247     case ulong:
248     {
249         {
250             var restrictionArray = stackalloc ulong[restrictionLength];
251             var ulongRestrictionArray = (IList<ulong>)restriction;
252             for (var i = 0; i < restrictionLength; i++)
253             {
254                 restrictionArray[i] = ulongRestrictionArray[i];
255             }
256             return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Count(_ptr,
257                               ↪ restrictionArray, (nuint)restrictionLength));
258         }
259     }
260     default:
261     {
262         throw new NotImplementedException();
263     }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }

public TLinkAddress Each(IList<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
    ↪ handler)
{
    var restrictionLength = restriction?.Count ?? 0;
    unsafe
    {
        TLinkAddress t = default;
        switch (t)
        {
            case byte:
            {
                byte Callback(FfiLink_UInt8 link) =>
                ↪ (byte)from_t.Convert(handler?.Invoke(new
                ↪ Link<TLinkAddress>(from_u8.Convert(link.Index),
                ↪ from_u8.Convert(link.Source), from_u8.Convert(link.Target))) ??
                ↪ Constants.Continue);
                var restrictionArray = stackalloc byte[restrictionLength];
                var byteRestrictionArray = (IList<byte>)restriction;
                for (var i = 0; i < restrictionLength; i++)
                {
                    restrictionArray[i] = byteRestrictionArray[i];
                }
                return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Each(_ptr,
                    ↪ restrictionArray, (nuint)restrictionLength, Callback));
            }
            case ushort:

```

```

285 {
286     ushort Callback(FfiLink_UInt16 link) =>
287         ↪ (ushort)from_t.Convert(handler?.Invoke(new
288         ↪ Link<TLinkAddress>(from_u16.Convert(link.Index),
289         ↪ from_u16.Convert(link.Source), from_u16.Convert(link.Target))) ??
290         ↪ Constants.Continue);
291     var restrictionArray = stackalloc ushort[restrictionLength];
292     var ushortRestrictionArray = (IList<ushort>)restriction;
293     for (var i = 0; i < restrictionLength; i++)
294     {
295         restrictionArray[i] = ushortRestrictionArray[i];
296     }
297     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Each(_ptr,
298         ↪ restrictionArray, (nuint)restrictionLength, Callback));
299 }
300 case uint:
301 {
302     uint Callback(FfiLink_UInt32 link) =>
303         ↪ (uint)from_t.Convert(handler?.Invoke(new
304         ↪ Link<TLinkAddress>(from_u32.Convert(link.Index),
305         ↪ from_u32.Convert(link.Source), from_u32.Convert(link.Target))) ??
306         ↪ Constants.Continue);
307     var restrictionArray = stackalloc uint[restrictionLength];
308     var uintRestrictionArray = (IList<uint>)restriction;
309     for (var i = 0; i < restrictionLength; i++)
310     {
311         restrictionArray[i] = uintRestrictionArray[i];
312     }
313     return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Each(_ptr,
314         ↪ restrictionArray, (nuint)restrictionLength, Callback));
315 }
316 case ulong:
317 {
318     {
319         ulong Callback(FfiLink_UInt64 link) =>
320             ↪ from_t.Convert(handler?.Invoke(new
321             ↪ Link<TLinkAddress>(from_u64.Convert(link.Index),
322             ↪ from_u64.Convert(link.Source), from_u64.Convert(link.Target)))
323             ↪ ?? Constants.Continue);
324         var restrictionArray = stackalloc UInt64[restrictionLength];
325         var ulongRestrictionArray = (IList<ulong>)restriction;
326         for (var i = 0; i < restrictionLength; i++)
327         {
328             restrictionArray[i] = ulongRestrictionArray[i];
329         }
330         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Each(_ptr,
331             ↪ restrictionArray, (nuint)restrictionLength, Callback));
332     }
333 }
334 default:
335 {
336     throw new NotImplementedException();
337 }
338 }
339 }
340 }
341
342 public TLinkAddress Create(IList<TLinkAddress>? substitution,
343     ↪ WriteHandler<TLinkAddress>? handler)
344 {
345     var substitutionLength = substitution?.Count ?? 0;
346     unsafe
347     {
348         TLinkAddress t = default;
349         switch (t)
350         {
351             case byte:
352             {
353                 byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
354                     ↪ (byte)from_t.Convert(handler?.Invoke(new
355                     ↪ Link<TLinkAddress>(from_u8.Convert(before.Index),
356                     ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
357                     ↪ Link<TLinkAddress>(from_u8.Convert(after.Index),
358                     ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) ??
359                     ↪ Constants.Continue);
360                 var substitutionArray = stackalloc byte[substitutionLength];
361                 var byteSubstitutionArray = (IList<byte>)substitution;
362                 for (var i = 0; i < substitutionLength; i++)

```

```

341         {
342             substitutionArray[i] = byteSubstitutionArray[i];
343         }
344         return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Create(_ptr,
345             ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
346     }
347     case ushort:
348     {
349         ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
350         (ushort)from_t.Convert(handler?.Invoke(new
351             ↳ Link<TLinkAddress>(from_u16.Convert(before.Index),
352             ↳ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
353             ↳ new Link<TLinkAddress>(from_u16.Convert(after.Index),
354             ↳ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) ??
355             ↳ Constants.Continue);
356         var substitutionArray = stackalloc ushort[substitutionLength];
357         var ushortSubstitutionArray = (IList<ushort>)substitution;
358         for (var i = 0; i < substitutionLength; i++)
359         {
360             substitutionArray[i] = ushortSubstitutionArray[i];
361         }
362         return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Create(_ptr,
363             ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
364     }
365     case uint:
366     {
367         uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
368         (uint)from_t.Convert(handler?.Invoke(new
369             ↳ Link<TLinkAddress>(from_u32.Convert(before.Index),
370             ↳ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
371             ↳ new Link<TLinkAddress>(from_u32.Convert(after.Index),
372             ↳ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) ??
373             ↳ Constants.Continue);
374         var substitutionArray = stackalloc uint[substitutionLength];
375         var uintSubstitutionArray = (IList<uint>)substitution;
376         for (var i = 0; i < substitutionLength; i++)
377         {
378             substitutionArray[i] = uintSubstitutionArray[i];
379         }
380         return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Create(_ptr,
381             ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
382     }
383     case ulong:
384     {
385         ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
386         (ulong)from_t.Convert(handler?.Invoke(new
387             ↳ Link<TLinkAddress>(from_u64.Convert(before.Index),
388             ↳ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
389             ↳ new Link<TLinkAddress>(from_u64.Convert(after.Index),
390             ↳ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) ??
391             ↳ Constants.Continue);
392         var substitutionArray = stackalloc ulong[substitutionLength];
393         var ulongSubstitutionArray = (IList<ulong>)substitution;
394         for (var i = 0; i < substitutionLength; i++)
395         {
396             substitutionArray[i] = ulongSubstitutionArray[i];
397         }
398         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Create(_ptr,
399             ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
400     }
401     default:
402     {
403         throw new NotImplementedException();
404     }
405 };
406 }
407 }
408
409 public TLinkAddress Update(IList<TLinkAddress>? restriction, IList<TLinkAddress>?
410     ↳ substitution, WriteHandler<TLinkAddress>? handler)
411 {
412     var restrictionLength = restriction?.Count ?? 0;
413     var substitutionLength = substitution?.Count ?? 0;
414     unsafe
415     {
416         TLinkAddress t = default;

```

```

396 switch (t)
397 {
398     case byte:
399     {
400         var restrictionArray = stackalloc byte[restrictionLength];
401         var byteRestrictionArray = (IList<byte>)restriction;
402         for (var i = 0; i < restrictionLength; i++)
403         {
404             restrictionArray[i] = byteRestrictionArray[i];
405         }
406         var substitutionArray = stackalloc byte[substitutionLength];
407         var byteSubstitutionArray = (IList<byte>)substitution;
408         for (var i = 0; i < substitutionLength; i++)
409         {
410             substitutionArray[i] = byteSubstitutionArray[i];
411         }
412         byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
413             (byte)from_t.Convert(handler?.Invoke(new
414                 ↳ Link<TLinkAddress>(from_u8.Convert(before.Index),
415                 ↳ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
416                 ↳ Link<TLinkAddress>(from_u8.Convert(after.Index),
417                 ↳ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) ??
418                 ↳ Constants.Continue);
419         return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Update(_ptr,
420             ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
421             ↳ (nuint)(substitution?.Count ?? 0), Callback));
422     }
423     case ushort:
424     {
425         var restrictionArray = stackalloc ushort[restrictionLength];
426         var ushortRestrictionArray = (IList<ushort>)restriction;
427         for (var i = 0; i < restrictionLength; i++)
428         {
429             restrictionArray[i] = ushortRestrictionArray[i];
430         }
431         var substitutionArray = stackalloc ushort[substitutionLength];
432         var ushortSubstitutionArray = (IList<ushort>)substitution;
433         for (var i = 0; i < substitutionLength; i++)
434         {
435             substitutionArray[i] = ushortSubstitutionArray[i];
436         }
437         ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
438             (ushort)from_t.Convert(handler?.Invoke(new
439                 ↳ Link<TLinkAddress>(from_u16.Convert(before.Index),
440                 ↳ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
441                 ↳ new Link<TLinkAddress>(from_u16.Convert(after.Index),
442                 ↳ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) ??
443                 ↳ Constants.Continue);
444         return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Update(_ptr,
445             ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
446             ↳ (nuint)(substitution?.Count ?? 0), Callback));
447     }
448     case uint:
449     {
450         var restrictionArray = stackalloc uint[restrictionLength];
451         var uintRestrictionArray = (IList<uint>)restriction;
452         for (var i = 0; i < restrictionLength; i++)
453         {
454             restrictionArray[i] = uintRestrictionArray[i];
455         }
456         var substitutionArray = stackalloc uint[substitutionLength];
457         var uintSubstitutionArray = (IList<uint>)substitution;
458         for (var i = 0; i < substitutionLength; i++)
459         {
460             substitutionArray[i] = uintSubstitutionArray[i];
461         }
462         uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
463             (uint)from_t.Convert(handler?.Invoke(new
464                 ↳ Link<TLinkAddress>(from_u32.Convert(before.Index),
465                 ↳ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
466                 ↳ new Link<TLinkAddress>(from_u32.Convert(after.Index),
467                 ↳ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) ??
468                 ↳ Constants.Continue);
469         return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Update(_ptr,
470             ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
471             ↳ (nuint)(substitution?.Count ?? 0), Callback));
472     }
473 }

```

```

449     case ulong:
450     {
451         var restrictionArray = stackalloc ulong[restrictionLength];
452         var ulongRestrictionArray = (IList<ulong>)restriction;
453         for (var i = 0; i < restrictionLength; i++)
454         {
455             restrictionArray[i] = ulongRestrictionArray[i];
456         }
457         var substitutionArray = stackalloc ulong[substitutionLength];
458         var ulongSubstitutionArray = (IList<ulong>)substitution;
459         for (var i = 0; i < substitutionLength; i++)
460         {
461             substitutionArray[i] = ulongSubstitutionArray[i];
462         }
463         ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
464             (ulong)from_t.Convert(handler?.Invoke(new
465                 ↳ Link<TLinkAddress>(from_u64.Convert(before.Index),
466                 ↳ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
467                 ↳ new Link<TLinkAddress>(from_u64.Convert(after.Index),
468                 ↳ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) ??
469                 ↳ Constants.Continue);
470         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Update(_ptr,
471             ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
472             ↳ (nuint)(substitution?.Count ?? 0), Callback));
473     }
474     default:
475     {
476         throw new NotImplementedException();
477     }
478 }
479 }
480 }
481
482 public TLinkAddress Delete(IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
483     ↳ handler)
484 {
485     var restrictionLength = restriction?.Count ?? 0;
486     unsafe
487     {
488         TLinkAddress t = default;
489         switch (t)
490         {
491             case byte:
492             {
493                 var restrictionArray = stackalloc byte[restrictionLength];
494                 var byteRestrictionArray = (IList<byte>)restriction;
495                 for (var i = 0; i < restrictionLength; i++)
496                 {
497                     restrictionArray[i] = byteRestrictionArray[i];
498                 }
499                 byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
500                     (byte)from_t.Convert(handler?.Invoke(new
501                         ↳ Link<TLinkAddress>(from_u8.Convert(before.Index),
502                         ↳ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
503                         ↳ Link<TLinkAddress>(from_u8.Convert(after.Index),
504                         ↳ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) ??
505                         ↳ Constants.Continue);
506                 return (TLinkAddress)(object)Methods.ByteUnitedMemoryLinks_Delete(_ptr,
507                     ↳ restrictionArray, (nuint)restrictionLength, Callback);
508             }
509             case ushort:
510             {
511                 var restrictionArray = stackalloc ushort[restrictionLength];
512                 var ushortRestrictionArray = (IList<ushort>)restriction;
513                 for (var i = 0; i < restrictionLength; i++)
514                 {
515                     restrictionArray[i] = ushortRestrictionArray[i];
516                 }
517                 ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
518                     (ushort)from_t.Convert(handler?.Invoke(new
519                         ↳ Link<TLinkAddress>(from_u16.Convert(before.Index),
520                         ↳ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
521                         ↳ new Link<TLinkAddress>(from_u16.Convert(after.Index),
522                         ↳ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) ??
523                         ↳ Constants.Continue);
524                 return
525                     (TLinkAddress)(object)Methods.UInt16UnitedMemoryLinks_Delete(_ptr,
526                     ↳ restrictionArray, (nuint)restrictionLength, Callback);

```

```

503     }
504     case uint:
505     {
506         var restrictionArray = stackalloc uint[restrictionLength];
507         var uintRestrictionArray = (IList<uint>)restriction;
508         for (var i = 0; i < restrictionLength; i++)
509         {
510             restrictionArray[i] = uintRestrictionArray[i];
511         }
512         uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
513         {
514             (uint)from_t.Convert(handler?.Invoke(new
515             Link<TLinkAddress>(from_u32.Convert(before.Index),
516             from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
517             new Link<TLinkAddress>(from_u32.Convert(after.Index),
518             from_u32.Convert(after.Source), from_u32.Convert(after.Target))) ??
519             Constants.Continue);
520             return
521             (TLinkAddress)(object)Methods.UInt32UnitedMemoryLinks_Delete(_ptr,
522             restrictionArray, (nuint)restrictionLength, Callback);
523         }
524     }
525     case ulong:
526     {
527         var restrictionArray = stackalloc ulong[restrictionLength];
528         var ulongRestrictionArray = (IList<ulong>)restriction;
529         for (var i = 0; i < restrictionLength; i++)
530         {
531             restrictionArray[i] = ulongRestrictionArray[i];
532         }
533         ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
534         {
535             (ulong)from_t.Convert(handler?.Invoke(new
536             Link<TLinkAddress>(from_u64.Convert(before.Index),
537             from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
538             new Link<TLinkAddress>(from_u64.Convert(after.Index),
539             from_u64.Convert(after.Source), from_u64.Convert(after.Target))) ??
540             Constants.Continue);
541             return
542             (TLinkAddress)(object)Methods.UInt64UnitedMemoryLinks_Delete(_ptr,
543             restrictionArray, (nuint)restrictionLength, Callback);
544         }
545     }
546     default:
547     {
548         throw new NotImplementedException();
549     }
550 }
551 }
552 }
553
554 protected override void Dispose(bool manual, bool wasDisposed)
555 {
556     unsafe
557     {
558         if (wasDisposed && _ptr != null)
559         {
560             return;
561         }
562         TLinkAddress t = default;
563         switch (t)
564         {
565             case byte:
566                 Methods.ByteUnitedMemoryLinks_Drop(_ptr);
567                 break;
568             case ushort:
569                 Methods.UInt16UnitedMemoryLinks_Drop(_ptr);
570                 break;
571             case uint:
572                 Methods.UInt32UnitedMemoryLinks_Drop(_ptr);
573                 break;
574             case ulong:
575                 Methods.UInt64UnitedMemoryLinks_Drop(_ptr);
576                 break;
577             default:
578                 throw new NotImplementedException();
579         }
580     }
581 }
582 }
583 }

```

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{TLinkAddress, LinksConstants{TLinkAddress}}"/>
14     public interface ILinks<TLinkAddress> : ILinks<TLinkAddress, LinksConstants<TLinkAddress>>
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="TLinkAddress">
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<TLinkAddress>(this ILinks<TLinkAddress> links,
48             ↪ ulong amountOfCreations) where TLinkAddress : struct
49         {
50             var random = RandomHelpers.Default;
51             var addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
52             var uint64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
53             for (var i = 0UL; i < amountOfCreations; i++)
54             {
55                 var linksAddressRange = new Range<ulong>(0,
56                     ↪ addressToUInt64Converter.Convert(links.Count()));
57                 var source =
58                     ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
59                 var target =
58                     ↪ 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="TLinkAddress">
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<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ ulong amountOfSearches) where TLinkAddress : struct
81 {
82     var random = RandomHelpers.Default;
83     var addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
84     var uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
85     for (var i = 0UL; i < amountOfSearches; i++)
86     {
87         var linksAddressRange = new Range<ulong>(0,
            ↪ addressToUInt64Converter.Convert(links.Count()));
88         var source =
            ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
89         var target =
            ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
90         links.SearchOrDefault(source, target);
91     }
92 }
93
94 /// <summary>
95 /// <para>
96 /// Runs the random deletions using the specified links.
97 /// </para>
98 /// <para></para>
99 /// </summary>
100 /// <typeparam name="TLinkAddress">
101 /// <para>The link.</para>
102 /// <para></para>
103 /// </typeparam>
104 /// <param name="links">
105 /// <para>The links.</para>
106 /// <para></para>
107 /// </param>
108 /// <param name="amountOfDeletions">
109 /// <para>The amount of deletions.</para>
110 /// <para></para>
111 /// </param>
112 [MethodImpl(MethodImplOptions.AggressiveInlining)]
113 public static void RunRandomDeletions<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ ulong amountOfDeletions) where TLinkAddress : struct
114 {
115     var random = RandomHelpers.Default;
116     var addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
117     var uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
118     var linksCount = addressToUInt64Converter.Convert(links.Count());
119     var min = amountOfDeletions > linksCount ? 0UL : linksCount - amountOfDeletions;
120     for (var i = 0UL; i < amountOfDeletions; i++)
121     {
122         linksCount = addressToUInt64Converter.Convert(links.Count());
123         if (linksCount <= min)
124         {
125             break;
126         }
127         var linksAddressRange = new Range<ulong>(min, linksCount);
128         var link =
            ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));

```

```

129         links.Delete(link);
130     }
131 }
132
133 /// <summary>
134 /// <para>
135 /// Deletes the links.
136 /// </para>
137 /// <para></para>
138 /// </summary>
139 /// <typeparam name="TLinkAddress">
140 /// <para>The link.</para>
141 /// <para></para>
142 /// </typeparam>
143 /// <param name="links">
144 /// <para>The links.</para>
145 /// <para></para>
146 /// </param>
147 /// <param name="linkToDelete">
148 /// <para>The link to delete.</para>
149 /// <para></para>
150 /// </param>
151 [MethodImpl(MethodImplOptions.AggressiveInlining)]
152 public static TLinkAddress Delete<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ TLinkAddress linkToDelete, WriteHandler<TLinkAddress>? handler) where TLinkAddress :
    ↪ struct
153 {
154     if (links.Exists(linkToDelete))
155     {
156         links.EnforceResetValues(linkToDelete, handler);
157     }
158     return links.Delete(new LinkAddress<TLinkAddress>(linkToDelete), handler);
159 }
160
161 /// <remarks>
162 /// TODO: Возможно есть очень простой способ это сделать.
163 /// (Например просто удалить файл, или изменить его размер таким образом,
164 /// чтобы удалился весь контент)
165 /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
166 /// </remarks>
167 [MethodImpl(MethodImplOptions.AggressiveInlining)]
168 public static void DeleteAll<TLinkAddress>(this ILinks<TLinkAddress> links) where
    ↪ TLinkAddress : struct
169 {
170     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
171     var comparer = Comparer<TLinkAddress>.Default;
172     for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
    ↪ Arithmetic.Decrement(i))
173     {
174         links.Delete(i);
175         if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
176         {
177             i = links.Count();
178         }
179     }
180 }
181
182 /// <summary>
183 /// <para>
184 /// Firsts the links.
185 /// </para>
186 /// <para></para>
187 /// </summary>
188 /// <typeparam name="TLinkAddress">
189 /// <para>The link.</para>
190 /// <para></para>
191 /// </typeparam>
192 /// <param name="links">
193 /// <para>The links.</para>
194 /// <para></para>
195 /// </param>
196 /// <exception cref="InvalidOperationException">
197 /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
198 /// <para></para>
199 /// </exception>
200 /// <exception cref="InvalidOperationException">
201 /// <para>В хранилище нет связей.</para>
202 /// <para></para>

```

```

203 /// </exception>
204 /// <returns>
205 /// <para>The first link.</para>
206 /// <para></para>
207 /// </returns>
208 [MethodImpl(MethodImplOptions.AggressiveInlining)]
209 public static TLinkAddress First<TLinkAddress>(this ILinks<TLinkAddress> links) where
    ↳ TLinkAddress : struct
210 {
211     TLinkAddress firstLink = default;
212     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
213     if (equalityComparer.Equals(links.Count(), default))
214     {
215         throw new InvalidOperationException("В хранилище нет связей.");
216     }
217     links.Each(links.Constants.Any, links.Constants.Any, link =>
218     {
219         firstLink = link[links.Constants.IndexPart];
220         return links.Constants.Break;
221     });
222     if (equalityComparer.Equals(firstLink, default))
223     {
224         throw new InvalidOperationException("В процессе поиска по хранилищу не было
            ↳ найдено связей.");
225     }
226     return firstLink;
227 }
228
229 /// <summary>
230 /// <para>
231 /// Singles the or default using the specified links.
232 /// </para>
233 /// <para></para>
234 /// </summary>
235 /// <typeparam name="TLinkAddress">
236 /// <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<TLinkAddress>? SingleOrDefault<TLinkAddress>(this
    ↳ ILinks<TLinkAddress> links, IList<TLinkAddress>? query) where TLinkAddress : struct
253 {
254     IList<TLinkAddress>? 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     TLinkAddress linkHandler(IList<TLinkAddress>? 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 /// <remarks>
280 /// TODO: Как так? Как то что ниже может быть корректно?
281 /// Скорее всего практически не применимо
282 /// Предполагалось, что можно было конвертировать формируемый в проходе через
283 → 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<TLinkAddress>(this ILinks<TLinkAddress> links,
289 → params TLinkAddress[] path) where TLinkAddress : struct
290 {
291     var current = path[0];
292     //EnsureLinkExists(current, "path");
293     if (!links.Exists(current))
294     {
295         return false;
296     }
297     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
298     var constants = links.Constants;
299     for (var i = 1; i < path.Length; i++)
300     {
301         var next = path[i];
302         var values = links.GetLink(current);
303         var source = values[constants.SourcePart];
304         var target = values[constants.TargetPart];
305         if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
306 → next))
307         {
308             //throw new InvalidOperationException(string.Format("Невозможно выбрать
309 → путь, так как и Source и Target совпадают с элементом пути {0}.", next));
310             return false;
311         }
312         if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
313 → target))
314         {
315             //throw new InvalidOperationException(string.Format("Невозможно продолжить
316 → путь через элемент пути {0}", next));
317             return false;
318         }
319         current = next;
320     }
321     return true;
322 }
323 /// <remarks>
324 /// Может потребовать дополнительного стека для PathElement's при использовании
325 → SequenceWalker.
326 /// </remarks>
327 [MethodImpl(MethodImplOptions.AggressiveInlining)]
328 public static TLinkAddress GetByKeys<TLinkAddress>(this ILinks<TLinkAddress> links,
329 → TLinkAddress root, params int[] path) where TLinkAddress : struct
330 {
331     links.EnsureLinkExists(root, "root");
332     var currentLink = root;
333     for (var i = 0; i < path.Length; i++)
334     {
335         currentLink = links.GetLink(currentLink)[path[i]];
336     }
337     return currentLink;
338 }
339 /// <summary>
340 /// <para>
341 /// Gets the square matrix sequence element by index using the specified links.
342 /// </para>
343 /// <para></para>
344 /// </summary>
345 /// <typeparam name="TLinkAddress">
346 /// <para>The link.</para>
347 /// <para></para>
348 /// </typeparam>
349 /// <param name="links">
350 /// <para>The links.</para>
351 /// <para></para>
352 /// </param>
353 /// <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 TLinkAddress GetSquareMatrixSequenceElementByIndex<TLinkAddress>(this
    ↳ ILinks<TLinkAddress> links, TLinkAddress root, ulong size, ulong index) where
    ↳ TLinkAddress : struct
370 {
371     var constants = links.Constants;
372     var source = constants.SourcePart;
373     var target = constants.TargetPart;
374     if (!Platform.Numbers.Math.IsPowerOfTwo(size))
375     {
376         throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
            ↳ than powers of two are not supported.");
377     }
378     var path = new BitArray(BitConverter.GetBytes(index));
379     var length = Bit.GetLowestPosition(size);
380     links.EnsureLinkExists(root, "root");
381     var currentLink = root;
382     for (var i = length - 1; i >= 0; i--)
383     {
384         currentLink = links.GetLink(currentLink)[path[i] ? target : source];
385     }
386     return currentLink;
387 }
388
389 #endregion
390
391 /// <summary>
392 /// Возвращает индекс указанной связи.
393 /// </summary>
394 /// <param name="links">Хранилище связей.</param>
395 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
396 /// <returns>Индекс начальной связи для указанной связи.</returns>
397 [MethodImpl(MethodImplOptions.AggressiveInlining)]
398 public static TLinkAddress GetIndex<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ IList<TLinkAddress>? link) where TLinkAddress : struct =>
    ↳ link[links.Constants.IndexPart];
399
400 /// <summary>
401 /// Возвращает индекс начальной (Source) связи для указанной связи.
402 /// </summary>
403 /// <param name="links">Хранилище связей.</param>
404 /// <param name="link">Индекс связи.</param>
405 /// <returns>Индекс начальной связи для указанной связи.</returns>
406 [MethodImpl(MethodImplOptions.AggressiveInlining)]
407 public static TLinkAddress GetSource<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress link) where TLinkAddress : struct =>
    ↳ links.GetLink(link)[links.Constants.SourcePart];
408
409 /// <summary>
410 /// Возвращает индекс начальной (Source) связи для указанной связи.
411 /// </summary>
412 /// <param name="links">Хранилище связей.</param>
413 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
414 /// <returns>Индекс начальной связи для указанной связи.</returns>
415 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

public static TLinkAddress GetSource<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ IList<TLinkAddress>? link) where TLinkAddress : struct =>
    ↳ link[links.Constants.SourcePart];

/// <summary>
/// Возвращает индекс конечной (Target) связи для указанной связи.
/// </summary>
/// <param name="links">Хранилище связей.</param>
/// <param name="link">Индекс связи.</param>
/// <returns>Индекс конечной связи для указанной связи.</returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static TLinkAddress GetTarget<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress link) where TLinkAddress : struct =>
    ↳ links.GetLink(link)[links.Constants.TargetPart];

/// <summary>
/// Возвращает индекс конечной (Target) связи для указанной связи.
/// </summary>
/// <param name="links">Хранилище связей.</param>
/// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
/// <returns>Индекс конечной связи для указанной связи.</returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static TLinkAddress GetTarget<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ IList<TLinkAddress>? link) where TLinkAddress : struct =>
    ↳ link[links.Constants.TargetPart];

/// <summary>
/// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
/// </summary>
/// <param name="links">Хранилище связей.</param>
/// <param name="handler">Обработчик каждой подходящей связи.</param>
/// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
    ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Any -
    ↳ отсутствие ограничения, 1..∞ конкретный адрес связи.</param>
/// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ ReadHandler<TLinkAddress>? handler, params TLinkAddress[] restriction) where
    ↳ TLinkAddress : struct
    => EqualityComparer<TLinkAddress>.Default.Equals(links.Each(restriction, handler),
        ↳ links.Constants.Continue);

public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ Func<TLinkAddress, bool> handler, TLinkAddress source, TLinkAddress target) where
    ↳ TLinkAddress : struct => links.Each(source, target, handler);

/// <summary>
/// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
/// </summary>
/// <param name="links">Хранилище связей.</param>
/// <param name="source">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
    ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
/// <param name="target">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
    ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
/// <param name="handler">Обработчик каждой подходящей связи.</param>
/// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
    ↳ source, TLinkAddress target, Func<TLinkAddress, bool> handler) where TLinkAddress :
    ↳ struct
{
    var constants = links.Constants;
    return links.Each(link => handler(link[constants.IndexPart]) ? constants.Continue :
        ↳ constants.Break, constants.Any, source, target);
}

public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ ReadHandler<TLinkAddress>? handler, TLinkAddress source, TLinkAddress target) where
    ↳ TLinkAddress : struct => links.Each(source, target, handler);

```

```

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

```

```

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

```



```

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

```

```

676 [MethodImpl(MethodImplOptions.AggressiveInlining)]
677 public static void EnsureLinkIsAnyOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↳ links, IList<TLinkAddress>? restriction) where TLinkAddress : struct
678 {
679     var equalityComparer = EqualityComparer<TLinkAddress>.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<TLinkAddress>(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="TLinkAddress">
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{TLinkAddress}">
713 /// <para></para>
714 /// <para></para>
715 /// </exception>
716 [MethodImpl(MethodImplOptions.AggressiveInlining)]
717 public static void EnsureLinkIsAnyOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↳ links, TLinkAddress link, string argumentName) where TLinkAddress : struct
718 {
719     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
720     if (!equalityComparer.Equals(link, links.Constants.Any) && !links.Exists(link))
721     {
722         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(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="TLinkAddress">
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{TLinkAddress}">
749 /// <para></para>

```

```

750 /// <para></para>
751 /// </exception>
752 [MethodImpl(MethodImplOptions.AggressiveInlining)]
753 public static void EnsureLinkIsItselfOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↳ links, TLinkAddress link, string argumentName) where TLinkAddress : struct
754 {
755     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
756     if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))
757     {
758         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
759     }
760 }
761
762 /// <param name="links">Хранилище связей.</param>
763 [MethodImpl(MethodImplOptions.AggressiveInlining)]
764 public static void EnsureDoesNotExists<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress source, TLinkAddress target) where TLinkAddress : struct
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<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress link) where TLinkAddress : struct
775 {
776     if (links.HasUsages(link))
777     {
778         throw new ArgumentLinkHasDependenciesException<TLinkAddress>(link);
779     }
780 }
781
782 /// <param name="links">Хранилище связей.</param>
783 [MethodImpl(MethodImplOptions.AggressiveInlining)]
784 public static void EnsureCreated<TLinkAddress>(this ILinks<TLinkAddress> links, params
    ↳ TLinkAddress[] addresses) where TLinkAddress : struct =>
    ↳ links.EnsureCreated(links.Create, addresses);
785
786 /// <param name="links">Хранилище связей.</param>
787 [MethodImpl(MethodImplOptions.AggressiveInlining)]
788 public static void EnsurePointsCreated<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ params TLinkAddress[] addresses) where TLinkAddress : struct =>
    ↳ links.EnsureCreated(links.CreatePoint, addresses);
789
790 /// <param name="links">Хранилище связей.</param>
791 [MethodImpl(MethodImplOptions.AggressiveInlining)]
792 public static void EnsureCreated<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ Func<TLinkAddress> creator, params TLinkAddress[] addresses) where TLinkAddress :
    ↳ struct
793 {
794     var addressToUInt64Converter = CheckedConverter<TLinkAddress, ulong>.Default;
795     var uInt64ToAddressConverter = CheckedConverter<ulong, TLinkAddress>.Default;
796     var nonExistentAddresses = new HashSet<TLinkAddress>(addresses.Where(x =>
    ↳ !links.Exists(x)));
797     if (nonExistentAddresses.Count > 0)
798     {
799         var max = nonExistentAddresses.Max();
800         max = uInt64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
    ↳ Convert(max),
    ↳ addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
    ↳ imum)));
801         var createdLinks = new List<TLinkAddress>();
802         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
803         TLinkAddress createdLink = creator();
804         while (!equalityComparer.Equals(createdLink, max))
805         {
806             createdLinks.Add(createdLink);
807         }
808         for (var i = 0; i < createdLinks.Count; i++)
809         {
810             if (!nonExistentAddresses.Contains(createdLinks[i]))
811             {
812                 links.Delete(createdLinks[i]);
813             }
814         }

```

```

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

```

```

880 public static TLinkAddress CreatePoint<TLinkAddress>(this ILinks<TLinkAddress> links,
881     ↳ WriteHandler<TLinkAddress>? handler) where TLinkAddress : struct
882 {
883     var constants = links.Constants;
884     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
885     ↳ constants.Break, handler);
886     TLinkAddress link = default;
887     TLinkAddress HandlerWrapper(ICollection<TLinkAddress>? before, ICollection<TLinkAddress>? after)
888     {
889         link = after[constants.IndexPart];
890         return handlerState.Handle(before, after);
891     }
892     handlerState.Apply(links.Create(null, HandlerWrapper));
893     handlerState.Apply(links.Update(link, link, link, HandlerWrapper));
894     return handlerState.Result;
895 }
896
897 public static TLinkAddress CreateAndUpdate<TLinkAddress>(this ILinks<TLinkAddress>
898     ↳ links, TLinkAddress source, TLinkAddress target) where TLinkAddress : struct
899 {
900     var constants = links.Constants;
901     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
902     ↳ constants.Break);
903     links.CreateAndUpdate(source, target, setter.SetFirstFromSecondListAndReturnTrue);
904     return setter.Result;
905 }
906
907 /// <param name="links">Хранилище связей.</param>
908 [MethodImpl(MethodImplOptions.AggressiveInlining)]
909 public static TLinkAddress CreateAndUpdate<TLinkAddress>(this ILinks<TLinkAddress>
910     ↳ links, TLinkAddress source, TLinkAddress target, WriteHandler<TLinkAddress>?
911     ↳ handler) where TLinkAddress : struct
912 {
913     var constants = links.Constants;
914     TLinkAddress createdLink = default;
915     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
916     ↳ constants.Break, handler);
917     handlerState.Apply(links.Create(null, (before, after) =>
918     {
919         createdLink = links.GetIndex(after);
920         return handlerState.Handle(before, after);
921     }));
922     handlerState.Apply(links.Update(createdLink, source, target, handler));
923     return handlerState.Result;
924 }
925
926 /// <summary>
927 /// Обновляет связь с указанными началом (Source) и концом (Target)
928 /// на связь с указанными началом (NewSource) и концом (NewTarget).
929 /// </summary>
930 /// <param name="links">Хранилище связей.</param>
931 /// <param name="link">Индекс обновляемой связи.</param>
932 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
933     ↳ выполняется обновление.</param>
934 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
935     ↳ выполняется обновление.</param>
936 /// <returns>Индекс обновлённой связи.</returns>
937 [MethodImpl(MethodImplOptions.AggressiveInlining)]
938 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
939     ↳ TLinkAddress link, TLinkAddress newSource, TLinkAddress newTarget) where
940     ↳ TLinkAddress : struct => links.Update(new LinkAddress<TLinkAddress>(link), new
941     ↳ Link<TLinkAddress>(link, newSource, newTarget));
942
943 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links, params
944     ↳ TLinkAddress[] restriction) where TLinkAddress : struct => links.Update(restriction,
945     ↳ null);
946
947 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
948     ↳ WriteHandler<TLinkAddress>? handler, params TLinkAddress[] restriction) where
949     ↳ TLinkAddress : struct => links.Update(restriction, handler);
950
951 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
952     ↳ ICollection<TLinkAddress>? restriction) where TLinkAddress : struct
953 {
954     var constants = links.Constants;
955     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
956     ↳ 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
951   ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Itself -
952   ↳ требование установить ссылку на себя, 1..∞ конкретный адрес другой связи.</param>
953 /// <returns>Индекс обновлённой связи.</returns>
954 [MethodImpl(MethodImplOptions.AggressiveInlining)]
955 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
956   ↳ IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>? handler) where
957   ↳ TLinkAddress : struct
958 {
959     return restriction.Count switch
960     {
961         2 => links.MergeAndDelete(restriction[0], restriction[1], handler),
962         4 => links.UpdateOrCreateOrGet(restriction[0], restriction[1], restriction[2],
963           ↳ restriction[3], handler),
964         _ => links.Update(restriction[0], restriction[1], restriction[2], handler)
965     };
966 }
967
968 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
969   ↳ TLinkAddress link, TLinkAddress newSource, TLinkAddress newTarget,
970   ↳ WriteHandler<TLinkAddress>? handler) where TLinkAddress : struct => links.Update(new
971   ↳ LinkAddress<TLinkAddress>(link), new Link<TLinkAddress>(link, newSource, newTarget),
972   ↳ handler);
973
974 /// <summary>
975 /// <para>
976 /// Resolves the constant as self reference using the specified links.
977 /// </para>
978 /// <para></para>
979 /// </summary>
980 /// <typeparam name="TLinkAddress">
981 /// <para>The link.</para>
982 /// <para></para>
983 /// </typeparam>
984 /// <param name="links">
985 /// <para>The links.</para>
986 /// <para></para>
987 /// </param>
988 /// <param name="constant">
989 /// <para>The constant.</para>
990 /// <para></para>
991 /// </param>
992 /// <param name="restriction">
993 /// <para>The restriction.</para>
994 /// <para></para>
995 /// </param>
996 /// <param name="substitution">
997 /// <para>The substitution.</para>
998 /// <para></para>
999 /// </param>
1000 /// <returns>
1001 /// <para>A list of t link</para>
1002 /// <para></para>
1003 /// </returns>
1004 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1005 public static IList<TLinkAddress>? ResolveConstantAsSelfReference<TLinkAddress>(this
1006   ↳ ILinks<TLinkAddress> links, TLinkAddress constant, IList<TLinkAddress>? restriction,
1007   ↳ IList<TLinkAddress>? substitution) where TLinkAddress : struct
1008 {
1009     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1010     var constants = links.Constants;
1011     var restrictionIndex = restriction[constants.IndexPart];
1012     var substitutionIndex = substitution[constants.IndexPart];
1013     if (equalityComparer.Equals(substitutionIndex, default))
1014     {
1015         substitutionIndex = restrictionIndex;
1016     }
1017     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<TLinkAddress>(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 TLinkAddress GetOrCreate<TLinkAddress>(this ILinks<TLinkAddress> links,
1025     ↪ TLinkAddress source, TLinkAddress target) where TLinkAddress : struct
1026     {
1027         var link = links.SearchOrDefault(source, target);
1028         if (EqualityComparer<TLinkAddress>.Default.Equals(link, default))
1029         {
1030             link = links.CreateAndUpdate(source, target);
1031         }
1032         return link;
1033     }
1034
1035     public static TLinkAddress UpdateOrCreateOrGet<TLinkAddress>(this ILinks<TLinkAddress>
1036     ↪ links, TLinkAddress source, TLinkAddress target, TLinkAddress newSource,
1037     ↪ TLinkAddress newTarget) where TLinkAddress : struct
1038     {
1039         var constants = links.Constants;
1040         var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
1041         ↪ constants.Break);
1042         links.UpdateOrCreateOrGet(source, target, newSource, newTarget,
1043         ↪ setter.SetFirstFromSecondListAndReturnTrue);
1044         return setter.Result;
1045     }
1046
1047     /// <summary>
1048     /// Обновляет связь с указанными началом (Source) и концом (Target)
1049     /// на связь с указанными началом (NewSource) и концом (NewTarget).
1050     /// </summary>
1051     /// <param name="links">Хранилище связей.</param>
1052     /// <param name="source">Индекс связи, которая является началом обновляемой
1053     /// связи.</param>
1054     /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
1055     /// <param name="newSource">Индекс связи, которая является началом связи, на которую
1056     /// выполняется обновление.</param>
1057     /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
1058     /// выполняется обновление.</param>
1059     /// <returns>Индекс обновлённой связи.</returns>
1060     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1061     public static TLinkAddress UpdateOrCreateOrGet<TLinkAddress>(this ILinks<TLinkAddress>
1062     ↪ links, TLinkAddress source, TLinkAddress target, TLinkAddress newSource,
1063     ↪ TLinkAddress newTarget, WriteHandler<TLinkAddress>? handler) where TLinkAddress :
1064     ↪ struct
1065     {
1066         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1067         var link = links.SearchOrDefault(source, target);
1068         if (equalityComparer.Equals(link, default))
1069         {
1070             return links.CreateAndUpdate(newSource, newTarget, handler);
1071         }
1072         if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
1073         ↪ target))
1074         {
1075             var linkStruct = new Link<TLinkAddress>(link, source, target);
1076             return link;
1077         }
1078         return links.Update(link, newSource, newTarget, handler);
1079     }
1080
1081     /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1082     /// <param name="links">Хранилище связей.</param>
1083     /// <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 TLinkAddress DeleteIfExists<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress source, TLinkAddress target) where TLinkAddress : struct
1072 {
1073     var link = links.SearchOrDefault(source, target);
1074     if (!EqualityComparer<TLinkAddress>.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<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ IList<TLinkAddress>? deletedLinks) where TLinkAddress : struct
1087 {
1088     for (int i = 0; i < deletedLinks.Count; i++)
1089     {
1090         links.Delete(deletedLinks[i]);
1091     }
1092 }
1093
1094 public static void DeleteAllUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress linkIndex) where TLinkAddress : struct =>
    ↳ 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 TLinkAddress DeleteAllUsages<TLinkAddress>(this ILinks<TLinkAddress>
    ↳ links, TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler) where
    ↳ TLinkAddress : struct
1099 {
1100     var constants = links.Constants;
1101     var any = constants.Any;
1102     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1103     var usagesAsSourceQuery = new Link<TLinkAddress>(any, linkIndex, any);
1104     var usagesAsTargetQuery = new Link<TLinkAddress>(any, any, linkIndex);
1105     var usages = new List<IList<TLinkAddress>?>();
1106     var usagesFiller = new ListFiller<IList<TLinkAddress>?, TLinkAddress>(usages,
        ↳ constants.Continue);
1107     links.Each(usagesFiller.AddAndReturnConstant, usagesAsSourceQuery);
1108     links.Each(usagesFiller.AddAndReturnConstant, usagesAsTargetQuery);
1109     WriteHandlerState<TLinkAddress> 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="TLinkAddress">
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<TLinkAddress>(this ILinks<TLinkAddress> links,
1141   ↳ Link<TLinkAddress> query) where TLinkAddress : struct
1142 {
1143     var queryResult = new List<TLinkAddress>();
1144     var queryResultFiller = new ListFiller<TLinkAddress, TLinkAddress>(queryResult,
1145   ↳ links.Constants.Continue);
1146     links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
1147     foreach (var link in queryResult)
1148     {
1149         links.Delete(link);
1150     }
1151 }
1152
1153 // TODO: Move to Platform.Data
1154 /// <summary>
1155 /// <para>
1156 /// Determines whether are values reset.
1157 /// </para>
1158 /// <para></para>
1159 /// </summary>
1160 /// <typeparam name="TLinkAddress">
1161 /// <para>The link.</para>
1162 /// <para></para>
1163 /// </typeparam>
1164 /// <param name="links">
1165 /// <para>The links.</para>
1166 /// <para></para>
1167 /// </param>
1168 /// <param name="linkIndex">
1169 /// <para>The link index.</para>
1170 /// <para></para>
1171 /// </param>
1172 /// <returns>
1173 /// <para>The bool</para>
1174 /// <para></para>
1175 /// </returns>
1176 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1177 public static bool AreValuesReset<TLinkAddress>(this ILinks<TLinkAddress> links,
1178   ↳ TLinkAddress linkIndex) where TLinkAddress : struct
1179 {
1180     var nullConstant = links.Constants.Null;
1181     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1182     var link = links.GetLink(linkIndex);
1183     for (int i = 1; i < link.Count; i++)
1184     {
1185         if (!equalityComparer.Equals(link[i], nullConstant))
1186         {
1187             return false;
1188         }
1189     }
1190     return true;
1191 }
1192
1193 public static void ResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1194   ↳ TLinkAddress linkIndex) where TLinkAddress : struct => links.ResetValues(linkIndex,
1195   ↳ null);
1196
1197 // TODO: Create a universal version of this method in Platform.Data (with using of for
1198   ↳ loop)
1199 /// <summary>
1200 /// <para>
1201 /// Resets the values using the specified links.
1202 /// </para>
1203 /// <para></para>
1204 /// </summary>
1205 /// <typeparam name="TLinkAddress">
1206 /// <para>The link.</para>
1207 /// <para></para>
1208 /// </typeparam>
1209 /// <param name="links">
1210 /// <para>The links.</para>
1211 /// <para></para>
1212 /// </param>
1213 /// <param name="linkIndex">

```

```

1208 /// <para>The link index.</para>
1209 /// <para></para>
1210 /// </param>
1211 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1212 public static TLinkAddress ResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler) where TLinkAddress :
    ↳ struct
1213 {
1214     var nullConstant = links.Constants.Null;
1215     var updateRequest = new Link<TLinkAddress>(linkIndex, nullConstant, nullConstant);
1216     return links.Update(updateRequest, handler);
1217 }
1218
1219 public static void EnforceResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress linkIndex) where TLinkAddress : struct =>
    ↳ links.EnforceResetValues(linkIndex, null);
1220
1221
1222 // TODO: Create a universal version of this method in Platform.Data (with using of for
    ↳ loop)
1223 /// <summary>
1224 /// <para>
1225 /// Enforces the reset values using the specified links.
1226 /// </para>
1227 /// <para></para>
1228 /// </summary>
1229 /// <typeparam name="TLinkAddress">
1230 /// <para>The link.</para>
1231 /// <para></para>
1232 /// </typeparam>
1233 /// <param name="links">
1234 /// <para>The links.</para>
1235 /// <para></para>
1236 /// </param>
1237 /// <param name="linkIndex">
1238 /// <para>The link index.</para>
1239 /// <para></para>
1240 /// </param>
1241 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1242 public static TLinkAddress EnforceResetValues<TLinkAddress>(this ILinks<TLinkAddress>
    ↳ links, TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler) where
    ↳ TLinkAddress : struct
1243 {
1244     if (!links.AreValuesReset(linkIndex))
1245     {
1246         return links.ResetValues(linkIndex, handler);
1247     }
1248     return links.Constants.Continue;
1249 }
1250
1251 public static void MergeUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex) where TLinkAddress : struct =>
    ↳ links.MergeUsages(oldLinkIndex, newLinkIndex, null);
1252
1253 /// <summary>
1254 /// Merging two usages graphs, all children of old link moved to be children of new link
    ↳ or deleted.
1255 /// </summary>
1256 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1257 public static TLinkAddress MergeUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex, WriteHandler<TLinkAddress>?
    ↳ handler) where TLinkAddress : struct
1258 {
1259     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1260     if (equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1261     {
1262         return newLinkIndex;
1263     }
1264     var constants = links.Constants;
1265     var usagesAsSource = links.All(new Link<TLinkAddress>(constants.Any, oldLinkIndex,
    ↳ constants.Any));
1266     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
    ↳ constants.Break, handler);
1267     for (var i = 0; i < usagesAsSource.Count; i++)
1268     {
1269         var usageAsSource = usagesAsSource[i];
1270         if (equalityComparer.Equals(usageAsSource[constants.IndexPart], oldLinkIndex))
1271         {

```

```

1272         continue;
1273     }
1274     var restriction = new
1275         ↪ LinkAddress<TLinkAddress>(usageAsSource[constants.IndexPart]);
1276     var substitution = new Link<TLinkAddress>(newLinkIndex,
1277         ↪ usageAsSource[constants.TargetPart]);
1278     handlerState.Apply(links.Update(restriction, substitution,
1279         ↪ handlerState.Handler));
1280 }
1281 var usagesAsTarget = links.All(new Link<TLinkAddress>(constants.Any, constants.Any,
1282     ↪ oldLinkIndex));
1283 for (var i = 0; i < usagesAsTarget.Count; i++)
1284 {
1285     var usageAsTarget = usagesAsTarget[i];
1286     if (equalityComparer.Equals(usageAsTarget[constants.IndexPart], oldLinkIndex))
1287     {
1288         continue;
1289     }
1290     var restriction = links.GetLink(usageAsTarget[constants.IndexPart]);
1291     var substitution = new Link<TLinkAddress>(usageAsTarget[constants.TargetPart],
1292         ↪ newLinkIndex);
1293     handlerState.Apply(links.Update(restriction, substitution,
1294         ↪ handlerState.Handler));
1295 }
1296 return handlerState.Result;
1297 }
1298
1299 public static TLinkAddress MergeAndDelete<TLinkAddress>(this ILinks<TLinkAddress> links,
1300     ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex) where TLinkAddress : struct
1301 {
1302     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1303     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1304     {
1305         links.MergeUsages(oldLinkIndex, newLinkIndex);
1306         links.Delete(oldLinkIndex);
1307     }
1308     return newLinkIndex;
1309 }
1310
1311 /// <summary>
1312 /// Replace one link with another (replaced link is deleted, children are updated or
1313     ↪ deleted).
1314 /// </summary>
1315 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1316 public static TLinkAddress MergeAndDelete<TLinkAddress>(this ILinks<TLinkAddress> links,
1317     ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex, WriteHandler<TLinkAddress>?
1318     ↪ handler) where TLinkAddress : struct
1319 {
1320     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1321     var constants = links.Constants;
1322     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
1323         ↪ constants.Break, handler);
1324     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1325     {
1326         handlerState.Apply(links.MergeUsages(oldLinkIndex, newLinkIndex,
1327             ↪ handlerState.Handler));
1328         handlerState.Apply(links.Delete(oldLinkIndex, handlerState.Handler));
1329     }
1330     return handlerState.Result;
1331 }
1332
1333 /// <summary>
1334 /// <para>
1335 /// Decorates the with automatic uniqueness and usages resolution using the specified
1336     ↪ links.
1337 /// </para>
1338 /// <para></para>
1339 /// </summary>
1340 /// <typeparam name="TLinkAddress">
1341 /// <para>The link.</para>
1342 /// <para></para>
1343 /// </typeparam>
1344 /// <param name="links">
1345 /// <para>The links.</para>
1346 /// <para></para>
1347 /// </param>
1348 /// <returns>
1349 /// <para>The links.</para>

```

```

1337     /// <para></para>
1338     /// </returns>
1339     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1340     public static ILinks<TLinkAddress>
1341     ↪ DecorateWithAutomaticUniquenessAndUsagesResolution<TLinkAddress>(this
1342     ↪ ILinks<TLinkAddress> links) where TLinkAddress : struct
1343     {
1344         links = new LinksCascadeUsagesResolver<TLinkAddress>(links);
1345         links = new NonNullContentsLinkDeletionResolver<TLinkAddress>(links);
1346         links = new LinksCascadeUniquenessAndUsagesResolver<TLinkAddress>(links);
1347         return links;
1348     }
1349
1350     /// <summary>
1351     /// <para>
1352     /// Formats the links.
1353     /// </para>
1354     /// <para></para>
1355     /// </summary>
1356     /// <typeparam name="TLinkAddress">
1357     /// <para>The link.</para>
1358     /// </typeparam>
1359     /// <param name="links">
1360     /// <para>The links.</para>
1361     /// </param>
1362     /// <param name="link">
1363     /// <para>The link.</para>
1364     /// </param>
1365     /// </returns>
1366     /// <para>The string</para>
1367     /// <para></para>
1368     /// </returns>
1369     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1370     public static string Format<TLinkAddress>(this ILinks<TLinkAddress> links,
1371     ↪ IList<TLinkAddress>? link) where TLinkAddress : struct
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="TLinkAddress">
1385     /// <para>The link.</para>
1386     /// </typeparam>
1387     /// <param name="links">
1388     /// <para>The links.</para>
1389     /// </param>
1390     /// <param name="link">
1391     /// <para>The link.</para>
1392     /// </param>
1393     /// </returns>
1394     /// <para>The string</para>
1395     /// <para></para>
1396     /// </returns>
1397     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1398     public static string Format<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
1399     ↪ link) where TLinkAddress : struct => links.Format(links.GetLink(link));
1400 }
1401 }
1402 }

```

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

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets
4  {
5      /// <summary>

```

```

6     /// <para>
7     /// Defines the synchronized links.
8     /// </para>
9     /// <para></para>
10    /// </summary>
11    /// <seealso cref="ISynchronizedLinks{TLinkAddress, ILinks{TLinkAddress}},
    ↳ LinksConstants{TLinkAddress}}"/>
12    /// <seealso cref="ILinks{TLinkAddress}"/>
13    public interface ISynchronizedLinks<TLinkAddress> : ISynchronizedLinks<TLinkAddress,
    ↳ ILinks<TLinkAddress>, LinksConstants<TLinkAddress>>, ILinks<TLinkAddress>
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<TLinkAddress> : IEquatable<Link<TLinkAddress>>,
    ↳ IReadOnlyList<TLinkAddress>, IList<TLinkAddress>
18     {
19         /// <summary>
20         /// <para>
21         /// The link.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public static readonly Link<TLinkAddress> Null = new Link<TLinkAddress>();
26         private static readonly LinksConstants<TLinkAddress> _constants =
    ↳ Default<LinksConstants<TLinkAddress>>.Instance;
27         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
    ↳ EqualityComparer<TLinkAddress>.Default;
28         private const int Length = 3;
29
30         /// <summary>
31         /// <para>
32         /// The index.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         public readonly TLinkAddress Index;
37         /// <summary>
38         /// <para>
39         /// The source.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public readonly TLinkAddress Source;
44         /// <summary>
45         /// <para>
46         /// The target.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         public readonly TLinkAddress Target;
51
52         /// <summary>
53         /// <para>
54         /// Initializes a new <see cref="Link"/> instance.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <param name="values">
59         /// <para>A values.</para>
60         /// <para></para>
61         /// </param>

```

```

62 [MethodImpl(MethodImplOptions.AggressiveInlining)]
63 public Link(params TLinkAddress[] values) => SetValues(values, out Index, out Source,
64     ↪ out Target);
65
66 /// <summary>
67 /// <para>
68 /// Initializes a new <see cref="Link"/> instance.
69 /// </para>
70 /// <para></para>
71 /// </summary>
72 /// <param name="values">
73 /// <para>A values.</para>
74 /// <para></para>
75 /// </param>
76 [MethodImpl(MethodImplOptions.AggressiveInlining)]
77 public Link(ICollection<TLinkAddress>? values) => SetValues(values, out Index, out Source, out
78     ↪ Target);
79
80 /// <summary>
81 /// <para>
82 /// Initializes a new <see cref="Link"/> instance.
83 /// </para>
84 /// <para></para>
85 /// </summary>
86 /// <param name="other">
87 /// <para>A other.</para>
88 /// <para></para>
89 /// </param>
90 /// <exception cref="NotSupportedException">
91 /// <para></para>
92 /// <para></para>
93 /// </exception>
94 [MethodImpl(MethodImplOptions.AggressiveInlining)]
95 public Link(object other)
96 {
97     if (other is Link<TLinkAddress> otherLink)
98     {
99         SetValues(ref otherLink, out Index, out Source, out Target);
100     }
101     else if (other is ICollection<TLinkAddress> otherList)
102     {
103         SetValues(otherList, out Index, out Source, out Target);
104     }
105     else
106     {
107         throw new NotSupportedException();
108     }
109 }
110
111 /// <summary>
112 /// <para>
113 /// Initializes a new <see cref="Link"/> instance.
114 /// </para>
115 /// <para></para>
116 /// </summary>
117 /// <param name="other">
118 /// <para>A other.</para>
119 /// <para></para>
120 /// </param>
121 [MethodImpl(MethodImplOptions.AggressiveInlining)]
122 public Link(ref Link<TLinkAddress> other) => SetValues(ref other, out Index, out Source,
123     ↪ out Target);
124
125 /// <summary>
126 /// <para>
127 /// Initializes a new <see cref="Link"/> instance.
128 /// </para>
129 /// <para></para>
130 /// </summary>
131 /// <param name="index">
132 /// <para>A index.</para>
133 /// <para></para>
134 /// </param>
135 /// <param name="source">
136 /// <para>A source.</para>
137 /// <para></para>
138 /// </param>
139 /// <param name="target">

```

```

137 /// <para>A target.</para>
138 /// <para></para>
139 /// </param>
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 public Link(TLinkAddress index, TLinkAddress source, TLinkAddress target)
142 {
143     Index = index;
144     Source = source;
145     Target = target;
146 }
147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
148 private static void SetValues(ref Link<TLinkAddress> other, out TLinkAddress index, out
    ↪ TLinkAddress source, out TLinkAddress target)
149 {
150     index = other.Index;
151     source = other.Source;
152     target = other.Target;
153 }
154 [MethodImpl(MethodImplOptions.AggressiveInlining)]
155 private static void SetValues(ICollection<TLinkAddress>? values, out TLinkAddress index, out
    ↪ TLinkAddress source, out TLinkAddress target)
156 {
157     if (values == null)
158     {
159         index = default;
160         source = default;
161         target = default;
162         return;
163     }
164     switch (values.Count)
165     {
166     case 3:
167         index = values[0];
168         source = values[1];
169         target = values[2];
170         break;
171     case 2:
172         index = values[0];
173         source = values[1];
174         target = default;
175         break;
176     case 1:
177         index = values[0];
178         source = default;
179         target = default;
180         break;
181     default:
182         index = default;
183         source = default;
184         target = default;
185         break;
186     }
187 }
188
189 /// <summary>
190 /// <para>
191 /// Gets the hash code.
192 /// </para>
193 /// <para></para>
194 /// </summary>
195 /// <returns>
196 /// <para>The int</para>
197 /// <para></para>
198 /// </returns>
199 [MethodImpl(MethodImplOptions.AggressiveInlining)]
200 public override int GetHashCode() => (Index, Source, Target).GetHashCode();
201
202 /// <summary>
203 /// <para>
204 /// Determines whether this instance is null.
205 /// </para>
206 /// <para></para>
207 /// </summary>
208 /// <returns>
209 /// <para>The bool</para>
210 /// <para></para>
211 /// </returns>
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
214     && _equalityComparer.Equals(Source, _constants.Null)

```

```

215         && _equalityComparer.Equals(Target, _constants.Null);
216
217     /// <summary>
218     /// <para>
219     /// Determines whether this instance equals.
220     /// </para>
221     /// <para></para>
222     /// </summary>
223     /// <param name="other">
224     /// <para>The other.</para>
225     /// <para></para>
226     /// </param>
227     /// <returns>
228     /// <para>The bool</para>
229     /// <para></para>
230     /// </returns>
231     [MethodImpl(MethodImplOptions.AggressiveInlining)]
232     public override bool Equals(object other) => other is Link<TLinkAddress> &&
        ↪ Equals((Link<TLinkAddress>)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<TLinkAddress> other) => _equalityComparer.Equals(Index,
        ↪ other.Index)
        && _equalityComparer.Equals(Source, other.Source)
        && _equalityComparer.Equals(Target, other.Target);
250
251
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(TLinkAddress index, TLinkAddress source, TLinkAddress
        ↪ 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(TLinkAddress source, TLinkAddress target) =>
298         ↪ $"({source}->{target})";
299
300     [MethodImpl(MethodImplOptions.AggressiveInlining)]
301     public static implicit operator TLinkAddress[] (Link<TLinkAddress> link) =>
302         ↪ link.ToArray();
303
304     [MethodImpl(MethodImplOptions.AggressiveInlining)]
305     public static implicit operator Link<TLinkAddress>(TLinkAddress[] linkArray) => new
306         ↪ Link<TLinkAddress>(linkArray);
307
308     /// <summary>
309     /// <para>
310     /// Returns the string.
311     /// </para>
312     /// <para></para>
313     /// </summary>
314     /// <returns>
315     /// <para>The string</para>
316     /// <para></para>
317     /// </returns>
318     [MethodImpl(MethodImplOptions.AggressiveInlining)]
319     public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
320         ↪ ToString(Source, Target) : ToString(Index, Source, Target);
321
322     #region IList
323
324     /// <summary>
325     /// <para>
326     /// Gets the count value.
327     /// </para>
328     /// <para></para>
329     /// </summary>
330     public int Count
331     {
332         [MethodImpl(MethodImplOptions.AggressiveInlining)]
333         get => Length;
334     }
335
336     /// <summary>
337     /// <para>
338     /// Gets the is read only value.
339     /// </para>
340     /// <para></para>
341     /// </summary>
342     public bool IsReadOnly
343     {
344         [MethodImpl(MethodImplOptions.AggressiveInlining)]
345         get => true;
346     }
347
348     /// <summary>
349     /// <para>
350     /// The not supported exception.
351     /// </para>
352     /// <para></para>
353     /// </summary>
354     public TLinkAddress this[int index]
355     {
356         [MethodImpl(MethodImplOptions.AggressiveInlining)]
357         get
358         {
359             Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
360                 ↪ nameof(index));
361             if (index == _constants.IndexPart)
362             {
363                 return Index;
364             }
365             if (index == _constants.SourcePart)
366             {
367                 return Source;
368             }
369         }
370     }

```

```

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<TLinkAddress> 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(TLinkAddress 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(TLinkAddress 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(TLinkAddress[] 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(TLinkAddress 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(TLinkAddress 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     }

```

```

517     }
518     if (_equalityComparer.Equals(Target, item))
519     {
520         return _constants.TargetPart;
521     }
522     return -1;
523 }
524
525 /// <summary>
526 /// <para>
527 /// Inserts the index.
528 /// </para>
529 /// <para></para>
530 /// </summary>
531 /// <param name="index">
532 /// <para>The index.</para>
533 /// <para></para>
534 /// </param>
535 /// <param name="item">
536 /// <para>The item.</para>
537 /// <para></para>
538 /// </param>
539 [MethodImpl(MethodImplOptions.AggressiveInlining)]
540 public void Insert(int index, TLinkAddress 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<TLinkAddress> left, Link<TLinkAddress> right) =>
557     ↪ left.Equals(right);
558
559 [MethodImpl(MethodImplOptions.AggressiveInlining)]
560 public static bool operator !=(Link<TLinkAddress> left, Link<TLinkAddress> right) =>
561     ↪ !(left == right);
562
563 #endregion
564 }
565 }

```

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="TLinkAddress">
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<TLinkAddress>(this Link<TLinkAddress> link) =>
35         ↪ Point<TLinkAddress>.IsFullPoint(link);
36
37     /// <summary>
38     /// <para>
39     /// Determines whether is partial point.
40     /// </para>
41     /// <para></para>
42     /// </summary>
43     /// <typeparam name="TLinkAddress">
44     /// <para>The link.</para>
45     /// <para></para>
46     /// </typeparam>
47     /// <param name="link">
48     /// <para>The link.</para>
49     /// <para></para>
50     /// </param>
51     /// <returns>
52     /// <para>The bool</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     public static bool IsPartialPoint<TLinkAddress>(this Link<TLinkAddress> link) =>
57         ↪ Point<TLinkAddress>.IsPartialPoint(link);
58 }
59 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links operator base.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public abstract class LinksOperatorBase<TLinkAddress>
14     {
15         /// <summary>
16         /// <para>
17         /// The links.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         protected readonly ILinks<TLinkAddress> _links;
22
23         /// <summary>
24         /// <para>
25         /// Gets the links value.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         public ILinks<TLinkAddress> 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<TLinkAddress> 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<TLinkAddress>
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(TLinkAddress 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(TLinkAddress 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<TLinkAddress>
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     TLinkAddress CountUsages(TLinkAddress 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     TLinkAddress Search(TLinkAddress source, TLinkAddress 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     TLinkAddress EachUsage(TLinkAddress root, ReadHandler<TLinkAddress>? 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 TLinkAddress root, TLinkAddress 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 TLinkAddress root, TLinkAddress 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<TLinkAddress> : IEquatable<LinksHeader<TLinkAddress>>
17     {
18         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
19             ↪ EqualityComparer<TLinkAddress>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<LinksHeader<TLinkAddress>>.Size;
28
29         /// <summary>
30         /// <para>

```



```

30     /// The allocated links.
31     /// </para>
32     /// <para></para>
33     /// </summary>
34     public TLinkAddress AllocatedLinks;
35     /// <summary>
36     /// <para>
37     /// The reserved links.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public TLinkAddress ReservedLinks;
42     /// <summary>
43     /// <para>
44     /// The free links.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     public TLinkAddress FreeLinks;
49     /// <summary>
50     /// <para>
51     /// The first free link.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     public TLinkAddress FirstFreeLink;
56     /// <summary>
57     /// <para>
58     /// The root as source.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLinkAddress RootAsSource;
63     /// <summary>
64     /// <para>
65     /// The root as target.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public TLinkAddress RootAsTarget;
70     /// <summary>
71     /// <para>
72     /// The last free link.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public TLinkAddress LastFreeLink;
77     /// <summary>
78     /// <para>
79     /// The reserved.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLinkAddress 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<TLinkAddress> linksHeader
101        ↪ ? Equals(linksHeader) : false;
102
103    /// <summary>
104    /// <para>
105    /// Determines whether this instance equals.
106    /// </para>
107    /// <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<TLinkAddress> 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<TLinkAddress> left, LinksHeader<TLinkAddress>
143     ↪ right) => left.Equals(right);
144
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static bool operator !=(LinksHeader<TLinkAddress> left, LinksHeader<TLinkAddress>
147     ↪ right) => !(left == right);
148 }
149 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.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{TLinkAddress}"/>
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class
23     ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress> :
24     ↪ RecursionlessSizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
25     {
26         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
27         ↪ = UncheckedConverter<TLinkAddress, long>.Default;
28
29         /// <summary>
30         /// <para>
31         /// The break.
32         /// </para>
33         /// <para></para>
34         /// </summary>

```

```

32     protected readonly TLinkAddress Break;
33     /// <summary>
34     /// <para>
35     /// The continue.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected readonly TLinkAddress 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="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
87     ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
88     ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
89     {
90         LinksDataParts = linksDataParts;
91         LinksIndexParts = linksIndexParts;
92         Header = header;
93         Break = constants.Break;
94         Continue = constants.Continue;
95     }
96
97     /// <summary>
98     /// <para>
99     /// Gets the tree root.
100    /// </para>
101    /// <para></para>
102    /// </summary>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected abstract TLinkAddress GetTreeRoot();
109
110    /// <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 TLinkAddress GetBasePartValue(TLinkAddress 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(TLinkAddress source, TLinkAddress
    ↪ target, TLinkAddress rootSource, TLinkAddress 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(TLinkAddress source, TLinkAddress
    ↪ target, TLinkAddress rootSource, TLinkAddress 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<TLinkAddress> GetHeaderReference() => ref
        ↳ AsRef<LinksHeader<TLinkAddress>>(Header);

194     /// <summary>
195     /// <para>
196     /// 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<TLinkAddress>
        ↳ GetLinkDataPartReference(TLinkAddress link) => ref
        ↳ AsRef<RawLinkDataPart<TLinkAddress>>(LinksDataParts +
        ↳ (RawLinkDataPart<TLinkAddress>.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<TLinkAddress>
        ↳ GetLinkIndexPartReference(TLinkAddress link) => ref
        ↳ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
        ↳ (RawLinkIndexPart<TLinkAddress>.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<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
244     {
245         ref var link = ref GetLinkDataPartReference(linkIndex);
246         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
247     }

248     /// <summary>
249     /// <para>
250     /// 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 FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
270     {
271         ref var firstLink = ref GetLinkDataPartReference(first);
272         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(TLinkAddress first, TLinkAddress
296         ↪ second)
297     {
298         ref var firstLink = ref GetLinkDataPartReference(first);
299         ref var secondLink = ref GetLinkDataPartReference(second);
300         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
301             ↪ secondLink.Source, secondLink.Target);
302     }
303     /// <summary>
304     /// <para>
305     /// The zero.
306     /// </para>
307     /// <para></para>
308     /// </summary>
309     public TLinkAddress this[TLinkAddress index]
310     {
311         [MethodImpl(MethodImplOptions.AggressiveInlining)]
312         get
313         {
314             var root = GetTreeRoot();
315             if (GreaterOrEqualThan(index, GetSize(root)))
316             {
317                 return Zero;
318             }
319             while (!EqualToZero(root))
320             {
321                 var left = GetLeftOrDefault(root);
322                 var leftSize = GetSizeOrZero(left);
323                 if (LessThan(index, leftSize))
324                 {
325                     root = left;
326                     continue;
327                 }
328                 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 TLinkAddress Search(TLinkAddress source, TLinkAddress target)
347 {
348     var root = GetTreeRoot();
349     while (!EqualToZero(root))
350     {
351         ref var rootLink = ref GetLinkDataPartReference(root);
352         var rootSource = rootLink.Source;
353         var rootTarget = rootLink.Target;
354         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
355             ↳ node.Key < root.Key
356         {
357             root = GetLeftOrDefault(root);
358         }
359         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
360             ↳ node.Key > root.Key
361         {
362             root = GetRightOrDefault(root);
363         }
364         else // node.Key == root.Key
365         {
366             return root;
367         }
368     }
369     return Zero;
370 }
371
372 // TODO: Return indices range instead of references count
373 /// <summary>
374 /// <para>
375 /// Counts the usages using the specified link.
376 /// </para>
377 /// <para></para>
378 /// </summary>
379 /// <param name="link">
380 /// <para>The link.</para>
381 /// <para></para>
382 /// </param>
383 /// <returns>
384 /// <para>The link</para>
385 /// <para></para>
386 /// </returns>
387 [MethodImpl(MethodImplOptions.AggressiveInlining)]
388 public TLinkAddress CountUsages(TLinkAddress link)
389 {
390     var root = GetTreeRoot();
391     var total = GetSize(root);
392     var totalRightIgnore = Zero;
393     while (!EqualToZero(root))
394     {
395         var @base = GetBasePartValue(root);
396         if (LessOrEqualThan(@base, link))
397         {
398             root = GetRightOrDefault(root);
399         }
400         else
401         {
402             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
403             root = GetLeftOrDefault(root);
404         }
405     }

```

```

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 TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
441     ↳ EachUsageCore(@base, GetTreeRoot(), handler);
442
443 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
444 ↳ low-level MSIL stack.
445 [MethodImpl(MethodImplOptions.AggressiveInlining)]
446 private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
447     ↳ ReadHandler<TLinkAddress>? handler)
448 {
449     var @continue = Continue;
450     if (EqualToZero(link))
451     {
452         return @continue;
453     }
454     var linkBasePart = GetBasePartValue(link);
455     var @break = Break;
456     if (GreaterThan(linkBasePart, @base))
457     {
458         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
459         {
460             return @break;
461         }
462     }
463     else if (LessThan(linkBasePart, @base))
464     {
465         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
466         {
467             return @break;
468         }
469     }
470     else //if (linkBasePart == @base)
471     {
472         if (AreEqual(handler(GetLinkValues(link)), @break))
473         {
474             return @break;
475         }
476         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
477         {
478             return @break;
479         }
480     }
481 }

```



```

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(TLinkAddress 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{TLinkAddress}" />
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}" />
22     public unsafe abstract class ExternalLinksSizeBalancedTreeMethodsBase<TLinkAddress> :
23     ↪ SizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
24     {
25         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
26         ↪ = UncheckedConverter<TLinkAddress, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLinkAddress Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLinkAddress Continue;

```

```

42     /// The links data parts.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     protected readonly byte* LinksDataParts;
47     /// <summary>
48     /// <para>
49     /// The links index parts.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     protected readonly byte* LinksIndexParts;
54     /// <summary>
55     /// <para>
56     /// The header.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     protected readonly byte* Header;
61
62     /// <summary>
63     /// <para>
64     /// Initializes a new <see cref="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<TLinkAddress>
86         ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
87     {
88         LinksDataParts = linksDataParts;
89         LinksIndexParts = linksIndexParts;
90         Header = header;
91         Break = constants.Break;
92         Continue = constants.Continue;
93     }
94     /// <summary>
95     /// <para>
96     /// Gets the tree root.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <returns>
101    /// <para>The link</para>
102    /// <para></para>
103    /// </returns>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    protected abstract TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress 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(TLinkAddress source, TLinkAddress
        ↪ target, TLinkAddress rootSource, TLinkAddress 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(TLinkAddress source, TLinkAddress
        ↪ target, TLinkAddress rootSource, TLinkAddress 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<TLinkAddress> GetHeaderReference() => ref
        ↪ AsRef<LinksHeader<TLinkAddress>>(Header);

```

```

194     /// <summary>
195     /// <para>
196     /// 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<TLinkAddress>
210     ↪ GetLinkDataPartReference(TLinkAddress link) => ref
211     ↪ AsRef<RawLinkDataPart<TLinkAddress>>(LinksDataParts +
212     ↪ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
213     ↪ _addressToInt64Converter.Convert(link)));
214
215     /// <summary>
216     /// <para>
217     /// Gets the link index part reference using the specified link.
218     /// </para>
219     /// <para></para>
220     /// </summary>
221     /// <param name="link">
222     /// <para>The link.</para>
223     /// <para></para>
224     /// </param>
225     /// <returns>
226     /// <para>A ref raw link index part of t link</para>
227     /// <para></para>
228     /// </returns>
229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
230     protected virtual ref RawLinkIndexPart<TLinkAddress>
231     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
232     ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
233     ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
234     ↪ _addressToInt64Converter.Convert(link)));
235
236     /// <summary>
237     /// <para>
238     /// Gets the link values using the specified link index.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="linkIndex">
243     /// <para>The link index.</para>
244     /// <para></para>
245     /// </param>
246     /// <returns>
247     /// <para>A list of t link</para>
248     /// <para></para>
249     /// </returns>
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
252     {
253         ref var link = ref GetLinkDataPartReference(linkIndex);
254         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
255     }
256
257     /// <summary>
258     /// <para>
259     /// Determines whether this instance first is to the left of second.
260     /// </para>
261     /// <para></para>
262     /// </summary>
263     /// <param name="first">
264     /// <para>The first.</para>
265     /// <para></para>
266     /// </param>
267     /// <param name="second">
268     /// <para>The second.</para>
269     /// <para></para>
270     /// </param>

```

```

264    /// <returns>
265    /// <para>The bool</para>
266    /// <para></para>
267    /// </returns>
268    [MethodImpl(MethodImplOptions.AggressiveInlining)]
269    protected override bool FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
270    {
271        ref var firstLink = ref GetLinkDataPartReference(first);
272        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(TLinkAddress first, TLinkAddress
296        ↪ second)
297    {
298        ref var firstLink = ref GetLinkDataPartReference(first);
299        ref var secondLink = ref GetLinkDataPartReference(second);
300        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
301            ↪ secondLink.Source, secondLink.Target);
302    }
303    /// <summary>
304    /// <para>
305    /// The zero.
306    /// </para>
307    /// <para></para>
308    /// </summary>
309    public TLinkAddress this[TLinkAddress index]
310    {
311        [MethodImpl(MethodImplOptions.AggressiveInlining)]
312        get
313        {
314            var root = GetTreeRoot();
315            if (GreaterOrEqualThan(index, GetSize(root)))
316            {
317                return Zero;
318            }
319            while (!EqualToZero(root))
320            {
321                var left = GetLeftOrDefault(root);
322                var leftSize = GetSizeOrZero(left);
323                if (LessThan(index, leftSize))
324                {
325                    root = left;
326                    continue;
327                }
328                if (AreEqual(index, leftSize))
329                {
330                    return root;
331                }
332                root = GetRightOrDefault(root);
333                index = Subtract(index, Increment(leftSize));
334            }
335            return Zero; // TODO: Impossible situation exception (only if tree structure
336            ↪ broken)
337        }
338    }

```

```

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 TLinkAddress Search(TLinkAddress source, TLinkAddress target)
347 {
348     var root = GetTreeRoot();
349     while (!EqualToZero(root))
350     {
351         ref var rootLink = ref GetLinkDataPartReference(root);
352         var rootSource = rootLink.Source;
353         var rootTarget = rootLink.Target;
354         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
355             → node.Key < root.Key
356         {
357             root = GetLeftOrDefault(root);
358         }
359         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
360             → node.Key > root.Key
361         {
362             root = GetRightOrDefault(root);
363         }
364         else // node.Key == root.Key
365         {
366             return root;
367         }
368     }
369     return Zero;
370 }
371
372 // TODO: Return indices range instead of references count
373 /// <summary>
374 /// <para>
375 /// Counts the usages using the specified link.
376 /// </para>
377 /// <para></para>
378 /// </summary>
379 /// <param name="link">
380 /// <para>The link.</para>
381 /// <para></para>
382 /// </param>
383 /// <returns>
384 /// <para>The link</para>
385 /// <para></para>
386 /// </returns>
387 [MethodImpl(MethodImplOptions.AggressiveInlining)]
388 public TLinkAddress CountUsages(TLinkAddress link)
389 {
390     var root = GetTreeRoot();
391     var total = GetSize(root);
392     var totalRightIgnore = Zero;
393     while (!EqualToZero(root))
394     {
395         var @base = GetBasePartValue(root);
396         if (LessOrEqualThan(@base, link))
397         {
398             root = GetRightOrDefault(root);
399         }
400         else
401         {
402             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
403             root = GetLeftOrDefault(root);
404         }
405     }
406     root = GetTreeRoot();
407     var totalLeftIgnore = Zero;
408     while (!EqualToZero(root))
409     {
410         var @base = GetBasePartValue(root);
411         if (GreaterOrEqualThan(@base, link))
412         {
413             root = GetLeftOrDefault(root);
414         }
415         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 TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
441     ↳ EachUsageCore(@base, GetTreeRoot(), handler);
442
443 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
444 ↳ low-level MSIL stack.
445 [MethodImpl(MethodImplOptions.AggressiveInlining)]
446 private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
447     ↳ ReadHandler<TLinkAddress>? handler)
448 {
449     var @continue = Continue;
450     if (EqualToZero(link))
451     {
452         return @continue;
453     }
454     var linkBasePart = GetBasePartValue(link);
455     var @break = Break;
456     if (GreaterThan(linkBasePart, @base))
457     {
458         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
459         {
460             return @break;
461         }
462     }
463     else if (LessThan(linkBasePart, @base))
464     {
465         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
466         {
467             return @break;
468         }
469     }
470     else //if (linkBasePart == @base)
471     {
472         if (AreEqual(handler(GetLinkValues(link)), @break))
473         {
474             return @break;
475         }
476         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
477         {
478             return @break;
479         }
480         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
481         {
482             return @break;
483         }
484     }
485     return @continue;
486 }
487
488 /// <summary>
489 /// <para>
490 /// 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(TLinkAddress 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{TLinkAddress}"/>
14     public unsafe class ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
15         ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
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<TLinkAddress> constants,
42             ↪ 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>

```



```

50     /// </param>
51     /// <returns>
52     /// <para>The ref link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress left) =>
125         ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
126
127     /// <summary>
128     /// <para>
129     /// Sets the right using the specified node.
130     /// </para>
131     /// </summary>
132     /// <param name="node">
133     /// <para>The node.</para>
134     /// </param>
135     /// <param name="right">
136     /// <para>The right.</para>
137     /// </param>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
140         ↪ GetLinkIndexPartReference(node).RightAsSource = right;
141
142     /// <summary>
143     /// <para>
144     /// Gets the size using the specified node.
145     /// </para>
146     /// </summary>
147     /// <param name="node">
148     /// <para>The node.</para>
149     /// </param>
150     /// <returns>
151     /// <para>The link</para>
152     /// </returns>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     protected override TLinkAddress GetSize(TLinkAddress node) =>
155         ↪ GetLinkIndexPartReference(node).SizeAsSource;
156
157     /// <summary>
158     /// <para>
159     /// Sets the size using the specified node.
160     /// </para>
161     /// </summary>
162     /// <param name="node">
163     /// <para>The node.</para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// </param>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
170         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// </summary>
177     /// <returns>
178     /// <para>The link</para>
179     /// </returns>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsSource;
182
183     /// <summary>
184     /// <para>
185     /// Gets the base part value using the specified link.
186     /// </para>
187     /// </summary>
188     /// <param name="link">
189     /// <para>The link.</para>
190

```

```

198     /// <para></para>
199     /// </param>
200     /// <returns>
201     /// <para>The link</para>
202     /// <para></para>
203     /// </returns>
204     [MethodImpl(MethodImplOptions.AggressiveInlining)]
205     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
206         ↪ GetLinkDataPartReference(link).Source;
207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance first is to the left of second.
211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <param name="firstSource">
215     /// <para>The first source.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
236         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
237         ↪ LessThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
238         ↪ LessThan(firstTarget, secondTarget));
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
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(TLinkAddress firstSource, TLinkAddress
268         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
269         ↪ GreaterThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
270         ↪ GreaterThan(firstTarget, secondTarget));
271
272     /// <summary>
273     /// <para>
274     /// 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(TLinkAddress node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);
279         link.LeftAsSource = Zero;
280         link.RightAsSource = Zero;
281         link.SizeAsSource = Zero;
282     }
283 }
284 }

```

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

```

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

```

```

56     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
57         ↪ GetLinkIndexPartReference(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 TLinkAddress GetRightReference(TLinkAddress node) => ref
75         ↪ GetLinkIndexPartReference(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 TLinkAddress GetLeft(TLinkAddress node) =>
93         ↪ GetLinkIndexPartReference(node).LeftAsSource;
94
95     /// <summary>
96     /// <para>
97     /// Gets the right using the specified node.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="node">
102    /// <para>The node.</para>
103    /// <para></para>
104    /// </param>
105    /// <returns>
106    /// <para>The link</para>
107    /// <para></para>
108    /// </returns>
109    [MethodImpl(MethodImplOptions.AggressiveInlining)]
110    protected override TLinkAddress GetRight(TLinkAddress node) =>
111        ↪ GetLinkIndexPartReference(node).RightAsSource;
112
113    /// <summary>
114    /// <para>
115    /// Sets the left using the specified node.
116    /// </para>
117    /// <para></para>
118    /// </summary>
119    /// <param name="node">
120    /// <para>The node.</para>
121    /// <para></para>
122    /// </param>
123    /// <param name="left">
124    /// <para>The left.</para>
125    /// <para></para>
126    /// </param>
127    [MethodImpl(MethodImplOptions.AggressiveInlining)]
128    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
129        ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
130
131    /// <summary>
132    /// <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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetSize(TLinkAddress 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(TLinkAddress node, TLinkAddress size) =>
178         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
179
180     /// <summary>
181     /// <para>
182     /// Gets the tree root.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <returns>
187     /// <para>The link</para>
188     /// <para></para>
189     /// </returns>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsSource;
192
193     /// <summary>
194     /// <para>
195     /// Gets the base part value using the specified link.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="link">
200     /// <para>The link.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The link</para>
205     /// <para></para>
206     /// </returns>

```

```

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

```

```

273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLinkAddress 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{TLinkAddress}"/>
14     public unsafe class ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
15         ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress> constants,
42             ↪ 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 TLinkAddress GetLeftReference(TLinkAddress node) => ref
61             ↪ GetLinkIndexPartReference(node).LeftAsTarget;
62
63         /// <summary>
64         /// <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 TLinkAddress GetRightReference(TLinkAddress node) => ref
74     ↪ GetLinkIndexPartReference(node).RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLinkAddress GetLeft(TLinkAddress node) =>
92     ↪ GetLinkIndexPartReference(node).LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLinkAddress GetRight(TLinkAddress node) =>
110    ↪ GetLinkIndexPartReference(node).RightAsTarget;
111
112    /// <summary>
113    /// <para>
114    /// Sets the left using the specified node.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="node">
119    /// <para>The node.</para>
120    /// <para></para>
121    /// </param>
122    /// <param name="left">
123    /// <para>The left.</para>
124    /// <para></para>
125    /// </param>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
128    ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>

```

```

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(TLinkAddress node, TLinkAddress 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></para>
152    /// </param>
153    /// <returns>
154    /// <para>The link</para>
155    /// <para></para>
156    /// </returns>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override TLinkAddress GetSize(TLinkAddress node) =>
159        ↪ GetLinkIndexPartReference(node).SizeAsTarget;
160
161    /// <summary>
162    /// <para>
163    /// Sets the size using the specified node.
164    /// </para>
165    /// </summary>
166    /// <param name="node">
167    /// <para>The node.</para>
168    /// <para></para>
169    /// </param>
170    /// <param name="size">
171    /// <para>The size.</para>
172    /// <para></para>
173    /// </param>
174    [MethodImpl(MethodImplOptions.AggressiveInlining)]
175    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
176        ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
177
178    /// <summary>
179    /// <para>
180    /// Gets the tree root.
181    /// </para>
182    /// </summary>
183    /// <returns>
184    /// <para>The link</para>
185    /// <para></para>
186    /// </returns>
187    [MethodImpl(MethodImplOptions.AggressiveInlining)]
188    protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsTarget;
189
190    /// <summary>
191    /// <para>
192    /// Gets the base part value using the specified link.
193    /// </para>
194    /// </summary>
195    /// <param name="link">
196    /// <para>The link.</para>
197    /// <para></para>
198    /// </param>
199    /// <returns>
200    /// <para>The link</para>
201    /// <para></para>
202    /// </returns>
203    [MethodImpl(MethodImplOptions.AggressiveInlining)]
204    protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
205        ↪ GetLinkDataPartReference(link).Target;
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 FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
    ↪ LessThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
    ↪ LessThan(firstSource, secondSource));

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(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress 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{TLinkAddress}"/>
14     public unsafe class ExternalLinksTargetsSizeBalancedTreeMethods<TLinkAddress> :
15         ↳ ExternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="ExternalLinksTargetsSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress node) => ref
78             ↳ GetLinkIndexPartReference(node).RightAsTarget;
79     }
80 }

```

```

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

```

```

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

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(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress 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{TLinkAddress}"/>
21     /// <seealso cref="TLinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class
23     ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress> :
24     ↪ RecursionlessSizeBalancedTreeMethods<TLinkAddress>, TLinksTreeMethods<TLinkAddress>
25     {
26         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
27         ↪ = UncheckedConverter<TLinkAddress, long>.Default;
28
29         /// <summary>
30         /// <para>
31         /// The break.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         protected readonly TLinkAddress Break;
36
37         /// <summary>
38         /// <para>
39         /// The continue.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         protected readonly TLinkAddress Continue;
44
45         /// <summary>
46         /// <para>
47         /// The links data parts.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         protected readonly byte* LinksDataParts;
52
53         /// <summary>
54         /// <para>
55         /// The links index parts.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         protected readonly byte* LinksIndexParts;
60
61         /// <summary>
62         /// <para>
63         /// The header.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         protected readonly byte* Header;
68
69         /// <summary>
70         /// <para>
71         /// Initializes a new <see
72         ↪ cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
73         /// </para>
74         /// <para></para>
75         /// </summary>
76         /// <param name="constants">
77         /// <para>A constants.</para>
78         /// <para></para>
79         /// </param>
80         /// <param name="linksDataParts">
81         /// <para>A links data parts.</para>
82         /// <para></para>
83         /// </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
86     ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
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 TLinkAddress GetTreeRoot(TLinkAddress link);
112
113    /// <summary>
114    /// <para>
115    /// Gets the base part value using the specified link.
116    /// </para>
117    /// <para></para>
118    /// </summary>
119    /// <param name="link">
120    /// <para>The link.</para>
121    /// <para></para>
122    /// </param>
123    /// <returns>
124    /// <para>The link</para>
125    /// <para></para>
126    /// </returns>
127    [MethodImpl(MethodImplOptions.AggressiveInlining)]
128    protected abstract TLinkAddress GetBasePartValue(TLinkAddress link);
129
130    /// <summary>
131    /// <para>
132    /// Gets the key part value using the specified link.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="link">
137    /// <para>The link.</para>
138    /// <para></para>
139    /// </param>
140    /// <returns>
141    /// <para>The link</para>
142    /// <para></para>
143    /// </returns>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected abstract TLinkAddress GetKeyPartValue(TLinkAddress link);
146
147    /// <summary>
148    /// <para>
149    /// Gets the link data part reference using the specified link.
150    /// </para>
151    /// <para></para>
152    /// </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<TLinkAddress>
161    ↪ GetLinkDataPartReference(TLinkAddress link) => ref
162    ↪ AsRef<RawLinkDataPart<TLinkAddress>>(LinksDataParts +
163    ↪ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
164    ↪ _addressToInt64Converter.Convert(link)));
165
166    /// <summary>
167    /// <para>
168    /// Gets the link index part reference using the specified link.
169    /// </para>
170    /// <para></para>
171    /// </summary>
172    /// <param name="link">
173    /// <para>The link.</para>
174    /// <para></para>
175    /// </param>
176    /// <returns>
177    /// <para>A ref raw link index part of t link</para>
178    /// <para></para>
179    /// </returns>
180    [MethodImpl(MethodImplOptions.AggressiveInlining)]
181    protected virtual ref RawLinkIndexPart<TLinkAddress>
182    ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
183    ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
184    ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
185    ↪ _addressToInt64Converter.Convert(link)));
186
187    /// <summary>
188    /// <para>
189    /// Determines whether this instance first is to the left of second.
190    /// </para>
191    /// <para></para>
192    /// </summary>
193    /// <param name="first">
194    /// <para>The first.</para>
195    /// <para></para>
196    /// </param>
197    /// <param name="second">
198    /// <para>The second.</para>
199    /// <para></para>
200    /// </param>
201    /// <returns>
202    /// <para>The bool</para>
203    /// <para></para>
204    /// </returns>
205    [MethodImpl(MethodImplOptions.AggressiveInlining)]
206    protected override bool FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress
207    ↪ second) => LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
208
209    /// <summary>
210    /// <para>
211    /// Determines whether this instance first is to the right of second.
212    /// </para>
213    /// <para></para>
214    /// </summary>
215    /// <param name="first">
216    /// <para>The first.</para>
217    /// <para></para>
218    /// </param>
219    /// <param name="second">
220    /// <para>The second.</para>
221    /// <para></para>
222    /// </param>
223    /// <returns>
224    /// <para>The bool</para>
225    /// <para></para>
226    /// </returns>
227    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

219     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
    ↪     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<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
237     {
238         ref var link = ref GetLinkDataPartReference(linkIndex);
239         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
240     }
241
242     /// <summary>
243     /// <para>
244     /// The zero.
245     /// </para>
246     /// <para></para>
247     /// </summary>
248     public TLinkAddress this[TLinkAddress link, TLinkAddress 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
    ↪             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 TLinkAddress Search(TLinkAddress source, TLinkAddress 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 TLinkAddress SearchCore(TLinkAddress root, TLinkAddress 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 TLinkAddress CountUsages(TLinkAddress 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 TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
365     ↳ EachUsageCore(@base, 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 TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
371     ↳ ReadHandler<TLinkAddress>? 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(TLinkAddress node, StringBuilder sb)
407 {
408     ref var link = ref GetLinkDataPartReference(node);
409     sb.Append(' ');
410     sb.Append(link.Source);
411     sb.Append('-');
412     sb.Append('>');
413     sb.Append(link.Target);
414 }
415 }
416 }

```

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

```

1 using System;
2 using System.Text;
3 using System.Collections.Generic;
4 using System.Runtime.CompilerServices;
5 using Platform.Collections.Methods.Trees;
6 using Platform.Converters;
7 using Platform.Delegates;
8 using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.Split.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the internal links size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizeBalancedTreeMethods{TLinkAddress}"/>
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class InternalLinksSizeBalancedTreeMethodsBase<TLinkAddress> :
23         ↳ SizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
24     {
25         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
26             ↳ = UncheckedConverter<TLinkAddress, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.

```

```

29     /// </para>
30     /// <para></para>
31     /// </summary>
32     protected readonly TLinkAddress Break;
33     /// <summary>
34     /// <para>
35     /// The continue.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected readonly TLinkAddress Continue;
40     /// <summary>
41     /// <para>
42     /// The links data parts.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     protected readonly byte* LinksDataParts;
47     /// <summary>
48     /// <para>
49     /// The links index parts.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     protected readonly byte* LinksIndexParts;
54     /// <summary>
55     /// <para>
56     /// The header.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     protected readonly byte* Header;
61
62     /// <summary>
63     /// <para>
64     /// Initializes a new <see cref="InternalLinksSizeBalancedTreeMethodsBase"/> instance.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="constants">
69     /// <para>A constants.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="linksDataParts">
73     /// <para>A links data parts.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="linksIndexParts">
77     /// <para>A links index parts.</para>
78     /// <para></para>
79     /// </param>
80     /// <param name="header">
81     /// <para>A header.</para>
82     /// <para></para>
83     /// </param>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
86     ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
87     {
88         LinksDataParts = linksDataParts;
89         LinksIndexParts = linksIndexParts;
90         Header = header;
91         Break = constants.Break;
92         Continue = constants.Continue;
93     }
94     /// <summary>
95     /// <para>
96     /// Gets the tree root 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 TLinkAddress GetTreeRoot(TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress 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 TLinkAddress GetKeyPartValue(TLinkAddress 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<TLinkAddress>
161    ↪ GetLinkDataPartReference(TLinkAddress link) => ref
162    ↪ AsRef<RawLinkDataPart<TLinkAddress>>(LinksDataParts +
163    ↪ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
164    ↪ _addressToInt64Converter.Convert(link)));
165
166    /// <summary>
167    /// <para>
168    /// Gets the link index part reference using the specified link.
169    /// </para>
170    /// <para></para>
171    /// </summary>
172    /// <param name="link">
173    /// <para>The link.</para>
174    /// <para></para>
175    /// </param>
176    /// <returns>
177    /// <para>A ref raw link index part of t link</para>
178    /// <para></para>
179    /// </returns>
180    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

177 protected virtual ref RawLinkIndexPart<TLinkAddress>
    ↳ GetLinkIndexPartReference(TLinkAddress link) => ref
    ↳ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
    ↳ (RawLinkIndexPart<TLinkAddress>.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(TLinkAddress first, TLinkAddress
    ↳ 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(TLinkAddress first, TLinkAddress
    ↳ 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<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
237 {
238     ref var link = ref GetLinkDataPartReference(linkIndex);
239     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
240 }
241
242 /// <summary>
243 /// <para>
244 /// The zero.
245 /// </para>
246 /// <para></para>
247 /// </summary>
248 public TLinkAddress this[TLinkAddress link, TLinkAddress 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 TLinkAddress Search(TLinkAddress source, TLinkAddress 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>
302 /// </param>
303 /// <returns>
304 /// <para>The zero.</para>
305 /// <para></para>
306 /// </returns>
307 [MethodImpl(MethodImplOptions.AggressiveInlining)]
308 protected TLinkAddress SearchCore(TLinkAddress root, TLinkAddress key)
309 {
310     while (!EqualToZero(root))
311     {
312         {
313             var rootKey = GetKeyPartValue(root);
314             if (LessThan(key, rootKey)) // node.Key < root.Key
315             {
316                 root = GetLeftOrDefault(root);
317             }
318             else if (GreaterThan(key, rootKey)) // node.Key > root.Key
319             {
320                 root = GetRightOrDefault(root);
321             }
322             else // node.Key == root.Key
323             {
324                 return root;
325             }
326         }
327     }
328     return Zero;
329 }

```

```

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 TLinkAddress CountUsages(TLinkAddress 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 TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
365     ↳ EachUsageCore(@base, 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 TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
371     ↳ ReadHandler<TLinkAddress>? handler)
372 {
373     var @continue = Continue;
374     if (EqualToZero(link))
375     {
376         return @continue;
377     }
378     var @break = Break;
379     if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
380     {
381         return @break;
382     }
383     if (AreEqual(handler(GetLinkValues(link)), @break))
384     {
385         return @break;
386     }
387     if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
388     {
389         return @break;
390     }
391     return @continue;
392 }
393
394 /// <summary>
395 /// <para>
396 /// Prints the node value using the specified node.
397 /// </para>
398 /// <para></para>
399 /// </summary>
400 /// <param name="node">
401 /// <para>The node.</para>
402 /// <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(TLinkAddress 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{TLinkAddress}"/>
20     public unsafe class InternalLinksSourcesLinkedListMethods<TLinkAddress> :
21         ↳ RelativeCircularDoublyLinkedListMethods<TLinkAddress>
22     {
23         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
24             ↳ = UncheckedConverter<TLinkAddress, 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 TLinkAddress Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLinkAddress Continue;
43
44         /// <summary>
45         /// <para>
46         /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="constants">
51         /// <para>A constants.</para>
52         /// <para></para>
53         /// </param>
54         /// <param name="linksDataParts">
55         /// <para>A links data parts.</para>
56         /// <para></para>
57         /// </param>
58         /// <param name="linksIndexParts">
59         /// <para>A links index parts.</para>
60         /// <para></para>
61         /// </param>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

59 public InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress> constants,
    ↳ byte* linksDataParts, byte* linksIndexParts)
60 {
61     _linksDataParts = linksDataParts;
62     _linksIndexParts = linksIndexParts;
63     Break = constants.Break;
64     Continue = constants.Continue;
65 }
66
67 /// <summary>
68 /// <para>
69 /// Gets the link data part reference using the specified link.
70 /// </para>
71 /// <para></para>
72 /// </summary>
73 /// <param name="link">
74 /// <para>The link.</para>
75 /// <para></para>
76 /// </param>
77 /// <returns>
78 /// <para>A ref raw link data part of t link</para>
79 /// <para></para>
80 /// </returns>
81 [MethodImpl(MethodImplOptions.AggressiveInlining)]
82 protected virtual ref RawLinkDataPart<TLinkAddress>
    ↳ GetLinkDataPartReference(TLinkAddress link) => ref
    ↳ AsRef<RawLinkDataPart<TLinkAddress>>(_linksDataParts +
    ↳ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));
83
84 /// <summary>
85 /// <para>
86 /// Gets the link index part reference using the specified link.
87 /// </para>
88 /// <para></para>
89 /// </summary>
90 /// <param name="link">
91 /// <para>The link.</para>
92 /// <para></para>
93 /// </param>
94 /// <returns>
95 /// <para>A ref raw link index part of t link</para>
96 /// <para></para>
97 /// </returns>
98 [MethodImpl(MethodImplOptions.AggressiveInlining)]
99 protected virtual ref RawLinkIndexPart<TLinkAddress>
    ↳ GetLinkIndexPartReference(TLinkAddress link) => ref
    ↳ AsRef<RawLinkIndexPart<TLinkAddress>>(_linksIndexParts +
    ↳ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));
100
101 /// <summary>
102 /// <para>
103 /// Gets the first using the specified head.
104 /// </para>
105 /// <para></para>
106 /// </summary>
107 /// <param name="head">
108 /// <para>The head.</para>
109 /// <para></para>
110 /// </param>
111 /// <returns>
112 /// <para>The link</para>
113 /// <para></para>
114 /// </returns>
115 [MethodImpl(MethodImplOptions.AggressiveInlining)]
116 protected override TLinkAddress GetFirst(TLinkAddress head) =>
    ↳ GetLinkIndexPartReference(head).RootAsSource;
117
118 /// <summary>
119 /// <para>
120 /// Gets the last using the specified head.
121 /// </para>
122 /// <para></para>
123 /// </summary>
124 /// <param name="head">
125 /// <para>The head.</para>
126 /// <para></para>

```

```

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

```

```

202     /// </summary>
203     /// <param name="head">
204     /// <para>The head.</para>
205     /// <para></para>
206     /// </param>
207     /// <param name="element">
208     /// <para>The element.</para>
209     /// <para></para>
210     /// </param>
211     [MethodImpl(MethodImplOptions.AggressiveInlining)]
212     protected override void SetFirst(TLinkAddress head, TLinkAddress element) =>
213         ↪ GetLinkIndexPartReference(head).RootAsSource = element;
214
215     /// <summary>
216     /// <para>
217     /// Sets the last using the specified head.
218     /// </para>
219     /// <para></para>
220     /// </summary>
221     /// <param name="head">
222     /// <para>The head.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="element">
226     /// <para>The element.</para>
227     /// <para></para>
228     /// </param>
229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
230     protected override void SetLast(TLinkAddress head, TLinkAddress element)
231     {
232         //var first = GetLinkIndexPartReference(head).RootAsSource;
233         //if (EqualToZero(first))
234         //{
235             //    SetFirst(head, element);
236         //}
237         //else
238         //{
239             //    SetPrevious(first, element);
240         //}
241     }
242
243     /// <summary>
244     /// <para>
245     /// Sets the previous using the specified element.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     /// <param name="element">
250     /// <para>The element.</para>
251     /// <para></para>
252     /// </param>
253     /// <param name="previous">
254     /// <para>The previous.</para>
255     /// <para></para>
256     /// </param>
257     [MethodImpl(MethodImplOptions.AggressiveInlining)]
258     protected override void SetPrevious(TLinkAddress element, TLinkAddress previous) =>
259         ↪ GetLinkIndexPartReference(element).LeftAsSource = previous;
260
261     /// <summary>
262     /// <para>
263     /// Sets the next using the specified element.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="element">
268     /// <para>The element.</para>
269     /// <para></para>
270     /// </param>
271     /// <param name="next">
272     /// <para>The next.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void SetNext(TLinkAddress element, TLinkAddress next) =>
277         ↪ GetLinkIndexPartReference(element).RightAsSource = next;
278
279     /// <summary>

```

```

277     /// <para>
278     /// Sets the size using the specified head.
279     /// </para>
280     /// <para></para>
281     /// </summary>
282     /// <param name="head">
283     /// <para>The head.</para>
284     /// <para></para>
285     /// </param>
286     /// <param name="size">
287     /// <para>The size.</para>
288     /// <para></para>
289     /// </param>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     protected override void SetSize(TLinkAddress head, TLinkAddress size) =>
292         ↪ GetLinkIndexPartReference(head).SizeAsSource = size;
293
294     /// <summary>
295     /// <para>
296     /// Counts the usages using the specified head.
297     /// </para>
298     /// </summary>
299     /// <param name="head">
300     /// <para>The head.</para>
301     /// <para></para>
302     /// </param>
303     /// <returns>
304     /// <para>The link</para>
305     /// <para></para>
306     /// </returns>
307     [MethodImpl(MethodImplOptions.AggressiveInlining)]
308     public TLinkAddress CountUsages(TLinkAddress head) => GetSize(head);
309
310     /// <summary>
311     /// <para>
312     /// Gets the link values using the specified link index.
313     /// </para>
314     /// <para></para>
315     /// </summary>
316     /// <param name="linkIndex">
317     /// <para>The link index.</para>
318     /// <para></para>
319     /// </param>
320     /// <returns>
321     /// <para>A list of t link</para>
322     /// <para></para>
323     /// </returns>
324     [MethodImpl(MethodImplOptions.AggressiveInlining)]
325     protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
326     {
327         ref var link = ref GetLinkDataPartReference(linkIndex);
328         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
329     }
330
331     /// <summary>
332     /// <para>
333     /// Eaches the usage using the specified source.
334     /// </para>
335     /// <para></para>
336     /// </summary>
337     /// <param name="source">
338     /// <para>The source.</para>
339     /// <para></para>
340     /// </param>
341     /// <param name="handler">
342     /// <para>The handler.</para>
343     /// <para></para>
344     /// </param>
345     /// <returns>
346     /// <para>The continue.</para>
347     /// <para></para>
348     /// </returns>
349     [MethodImpl(MethodImplOptions.AggressiveInlining)]
350     public TLinkAddress EachUsage(TLinkAddress source, ReadHandler<TLinkAddress>? 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{TLinkAddress}"/>
14     public unsafe class InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
15         ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddr_
42         ↪ ess> 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>

```



```

55 [MethodImpl(MethodImplOptions.AggressiveInlining)]
56 protected override ref TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetSize(TLinkAddress 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetTreeRoot(TLinkAddress 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">

```

```

201     /// <para>The link.</para>
202     /// <para></para>
203     /// </param>
204     /// <returns>
205     /// <para>The link</para>
206     /// <para></para>
207     /// </returns>
208     [MethodImpl(MethodImplOptions.AggressiveInlining)]
209     protected override TLinkAddress GetBasePartValue(TLinkAddress 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 TLinkAddress GetKeyPartValue(TLinkAddress 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(TLinkAddress 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 TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
268         ↪ SearchCore(GetTreeRoot(source), target);
269 }

```

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

```

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

```

```

7  /// <summary>
8  /// <para>
9  /// Represents the internal links sources size balanced tree methods.
10 /// </para>
11 /// <para></para>
12 /// </summary>
13 /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLinkAddress}"/>
14 public unsafe class InternalLinksSourcesSizeBalancedTreeMethods<TLinkAddress> :
    ↳ InternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>
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<TLinkAddress>
        ↳ 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress 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>

```

```

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

```

```

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

```

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

```

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

```

```

33     /// </param>
34     /// <param name="header">
35     /// <para>A header.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddr_
    ↪     ess> 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetSize(TLinkAddress 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)]
179     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
180         ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;

```

```

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 TLinkAddress GetTreeRoot(TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress 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 TLinkAddress GetKeyPartValue(TLinkAddress 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(TLinkAddress 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>

```

```

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 TLinkAddress Search(TLinkAddress source, TLinkAddress 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{TLinkAddress}"/>
14     public unsafe class InternalLinksTargetsSizeBalancedTreeMethods<TLinkAddress> :
        ↪ InternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="InternalLinksTargetsSizeBalancedTreeMethods"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="constants">
23         /// <para>A constants.</para>
24         /// <para></para>
25         /// </param>
26         /// <param name="linksDataParts">
27         /// <para>A links data parts.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="linksIndexParts">
31         /// <para>A links index parts.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="header">
35         /// <para>A header.</para>
36         /// <para></para>
37         /// </param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
            ↪ 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 TLinkAddress GetLeftReference(TLinkAddress node) => ref
57         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// </summary>
64     /// <param name="node">
65     /// <para>The node.</para>
66     /// </param>
67     /// </returns>
68     /// <para>The ref link</para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
72         ↪ GetLinkIndexPartReference(node).RightAsTarget;
73
74     /// <summary>
75     /// <para>
76     /// Gets the left using the specified node.
77     /// </para>
78     /// </summary>
79     /// <param name="node">
80     /// <para>The node.</para>
81     /// </param>
82     /// </returns>
83     /// <para>The link</para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLinkAddress GetLeft(TLinkAddress node) =>
87         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
88
89     /// <summary>
90     /// <para>
91     /// Gets the right using the specified node.
92     /// </para>
93     /// </summary>
94     /// <param name="node">
95     /// <para>The node.</para>
96     /// </param>
97     /// </returns>
98     /// <para>The link</para>
99     /// </returns>
100     [MethodImpl(MethodImplOptions.AggressiveInlining)]
101     protected override TLinkAddress GetRight(TLinkAddress node) =>
102         ↪ GetLinkIndexPartReference(node).RightAsTarget;
103
104     /// <summary>
105     /// <para>
106     /// Sets the left using the specified node.
107     /// </para>
108     /// </summary>
109     /// <param name="node">
110     /// <para>The node.</para>
111     /// </param>
112     /// <param name="left">
113     /// <para>The left.</para>
114     /// </param>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
117         ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
118
119     /// <summary>
120     /// <para>
121     ///
122     /// </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(TLinkAddress node, TLinkAddress right) =>
142         ↪ GetLinkIndexPartReference(node).RightAsTarget = right;
143
144     /// <summary>
145     /// <para>
146     /// Gets the size using the specified node.
147     /// </para>
148     /// <para></para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override TLinkAddress GetSize(TLinkAddress node) =>
160         ↪ GetLinkIndexPartReference(node).SizeAsTarget;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="node">
169     /// <para>The node.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="size">
173     /// <para>The size.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
178         ↪ GetLinkIndexPartReference(node).SizeAsTarget = 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 TLinkAddress GetTreeRoot(TLinkAddress link) =>
196         ↪ GetLinkIndexPartReference(link).RootAsTarget;
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>

```

```

202     /// <para></para>
203     /// </param>
204     /// <returns>
205     /// <para>The link</para>
206     /// <para></para>
207     /// </returns>
208     [MethodImpl(MethodImplOptions.AggressiveInlining)]
209     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
210         ↪ GetLinkDataPartReference(link).Target;
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 TLinkAddress GetKeyPartValue(TLinkAddress link) =>
228         ↪ GetLinkDataPartReference(link).Source;
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(TLinkAddress node)
242     {
243         ref var link = ref GetLinkIndexPartReference(node);
244         link.LeftAsTarget = Zero;
245         link.RightAsTarget = Zero;
246         link.SizeAsTarget = 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 TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
268         ↪ SearchCore(GetTreeRoot(target), source);
269 }

```

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

```

```

81 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
    ↳ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
    ↳ IndexTreeType.Default, useLinkedList: true) { }
82
83 /// <summary>
84 /// <para>
85 /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
86 /// </para>
87 /// </summary>
88 /// <param name="dataMemory">
89 /// <para>A data memory.</para>
90 /// </param>
91 /// <param name="indexMemory">
92 /// <para>A index memory.</para>
93 /// </param>
94 /// <param name="memoryReservationStep">
95 /// <para>A memory reservation step.</para>
96 /// </param>
97 /// <param name="constants">
98 /// <para>A constants.</para>
99 /// </param>
100 [MethodImpl(MethodImplOptions.AggressiveInlining)]
101 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants) :
    ↳ this(dataMemory, indexMemory, memoryReservationStep, constants,
    ↳ IndexTreeType.Default, useLinkedList: true) { }
102
103 /// <summary>
104 /// <para>
105 /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
106 /// </para>
107 /// </summary>
108 /// <param name="dataMemory">
109 /// <para>A data memory.</para>
110 /// </param>
111 /// <param name="indexMemory">
112 /// <para>A index memory.</para>
113 /// </param>
114 /// <param name="memoryReservationStep">
115 /// <para>A memory reservation step.</para>
116 /// </param>
117 /// <param name="constants">
118 /// <para>A constants.</para>
119 /// </param>
120 /// <param name="indexTreeType">
121 /// <para>A index tree type.</para>
122 /// </param>
123 /// <param name="useLinkedList">
124 /// <para>A use linked list.</para>
125 /// </param>
126 [MethodImpl(MethodImplOptions.AggressiveInlining)]
127 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants,
    ↳ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
    ↳ memoryReservationStep, constants, useLinkedList)
128 {
129     if (indexTreeType == IndexTreeType.SizeBalancedTree)
130     {
131         _createInternalSourceTreeMethods = () => new
            ↳ InternalLinksSourcesSizeBalancedTreeMethods<TLinkAddress>(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
132         _createExternalSourceTreeMethods = () => new
            ↳ ExternalLinksSourcesSizeBalancedTreeMethods<TLinkAddress>(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
133     }
134 }
135
136
137
138
139
140
141
142
143
144

```



```

145         _createInternalTargetTreeMethods = () => new
            ↳ InternalLinksTargetsSizeBalancedTreeMethods<TLinkAddress>(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
146         _createExternalTargetTreeMethods = () => new
            ↳ ExternalLinksTargetsSizeBalancedTreeMethods<TLinkAddress>(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
147     }
148     else
149     {
150         _createInternalSourceTreeMethods = () => new InternalLinksSourcesRecursionlessSi
            ↳ zeBalancedTreeMethods<TLinkAddress>(Constants, _linksDataParts,
            ↳ _linksIndexParts, _header);
151         _createExternalSourceTreeMethods = () => new ExternalLinksSourcesRecursionlessSi
            ↳ zeBalancedTreeMethods<TLinkAddress>(Constants, _linksDataParts,
            ↳ _linksIndexParts, _header);
152         _createInternalTargetTreeMethods = () => new InternalLinksTargetsRecursionlessSi
            ↳ zeBalancedTreeMethods<TLinkAddress>(Constants, _linksDataParts,
            ↳ _linksIndexParts, _header);
153         _createExternalTargetTreeMethods = () => new ExternalLinksTargetsRecursionlessSi
            ↳ zeBalancedTreeMethods<TLinkAddress>(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<TLinkAddress>(Constants,
            ↳ _linksDataParts, _linksIndexParts);
181     }
182     else
183     {
184         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
185     }
186     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
187     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
188     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
189     UnusedLinksListMethods = new UnusedLinksListMethods<TLinkAddress>(_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<TLinkAddress> GetHeaderReference() => ref
        ↳ AsRef<LinksHeader<TLinkAddress>>(_header);
219
220     /// <summary>
221     /// <para>
222     /// Gets the link data part reference using the specified link index.
223     /// </para>
224     /// <para></para>
225     /// </summary>
226     /// <param name="linkIndex">
227     /// <para>The link index.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>A ref raw link data part of t link</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override ref RawLinkDataPart<TLinkAddress>
        ↳ GetLinkDataPartReference(TLinkAddress linkIndex) => ref
        ↳ AsRef<RawLinkDataPart<TLinkAddress>>(_linksDataParts + (LinkDataPartSizeInBytes *
        ↳ ConvertToInt64(linkIndex)));
236
237     /// <summary>
238     /// <para>
239     /// Gets the link index part reference using the specified link index.
240     /// </para>
241     /// <para></para>
242     /// </summary>
243     /// <param name="linkIndex">
244     /// <para>The link index.</para>
245     /// <para></para>
246     /// </param>
247     /// <returns>
248     /// <para>A ref raw link index part of t link</para>
249     /// <para></para>
250     /// </returns>
251     [MethodImpl(MethodImplOptions.AggressiveInlining)]
252     protected override ref RawLinkIndexPart<TLinkAddress>
        ↳ GetLinkIndexPartReference(TLinkAddress linkIndex) => ref
        ↳ AsRef<RawLinkIndexPart<TLinkAddress>>(_linksIndexParts + (LinkIndexPartSizeInBytes *
        ↳ ConvertToInt64(linkIndex)));
253 }
254 }

```

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="TLinks{TLinkAddress}"/>
24  public abstract class SplitMemoryLinksBase<TLinkAddress> : DisposableBase,
    ↳ ILinks<TLinkAddress> where TLinkAddress : struct
25  {
26      private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
    ↳ EqualityComparer<TLinkAddress>.Default;
27      private static readonly Comparer<TLinkAddress> _comparer =
    ↳ Comparer<TLinkAddress>.Default;
28      private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
    ↳ = UncheckedConverter<TLinkAddress, long>.Default;
29      private static readonly UncheckedConverter<long, TLinkAddress> _int64ToAddressConverter
    ↳ = UncheckedConverter<long, TLinkAddress>.Default;
30      private static readonly TLinkAddress _zero = default;
31      private static readonly TLinkAddress _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 LinkDataPartSizeInBytes =
    ↳ RawLinkDataPart<TLinkAddress>.SizeInBytes;
39
40      /// <summary>
41      /// <para>
42      /// The size in bytes.
43      /// </para>
44      /// <para></para>
45      /// </summary>
46      public static readonly long LinkIndexPartSizeInBytes =
    ↳ RawLinkIndexPart<TLinkAddress>.SizeInBytes;
47
48      /// <summary>
49      /// <para>
50      /// The size in bytes.
51      /// </para>
52      /// <para></para>
53      /// </summary>
54      public static readonly long LinkHeaderSizeInBytes =
    ↳ LinksHeader<TLinkAddress>.SizeInBytes;
55
56      /// <summary>
57      /// <para>
58      /// The default links size step.
59      /// </para>
60      /// <para></para>
61      /// </summary>
62      public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
63
64      /// <summary>
65      /// <para>
66      /// The data memory.
67      /// </para>
68      /// <para></para>
69      /// </summary>
70      protected readonly IResizableDirectMemory _dataMemory;
71      /// <summary>
72      /// <para>
73      /// The index memory.
74      /// </para>
75      /// <para></para>
76      /// </summary>
77      protected readonly IResizableDirectMemory _indexMemory;
78      /// <summary>
79      /// <para>
80      /// The use linked list.
81      /// </para>
82      /// <para></para>
83      /// </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<TLinkAddress> InternalSourcesListMethods;
107    /// <summary>
108    /// <para>
109    /// The internal sources tree methods.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    protected ILinksTreeMethods<TLinkAddress> InternalSourcesTreeMethods;
114    /// <summary>
115    /// <para>
116    /// The external sources tree methods.
117    /// </para>
118    /// <para></para>
119    /// </summary>
120    protected ILinksTreeMethods<TLinkAddress> ExternalSourcesTreeMethods;
121    /// <summary>
122    /// <para>
123    /// The internal targets tree methods.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    protected ILinksTreeMethods<TLinkAddress> InternalTargetsTreeMethods;
128    /// <summary>
129    /// <para>
130    /// The external targets tree methods.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    protected ILinksTreeMethods<TLinkAddress> 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<TLinkAddress> UnusedLinksListMethods;
145
146    /// <summary>
147    /// Возвращает общее число связей находящихся в хранилище.
148    /// </summary>
149    protected virtual TLinkAddress Total
150    {
151        [MethodImpl(MethodImplOptions.AggressiveInlining)]
152        get
153        {
154            ref var header = ref GetHeaderReference();
155            return Subtract(header.AllocatedLinks, header.FreeLinks);
156        }
157    }
158
159    /// <summary>
160    /// <para>
161    /// Gets the constants value.
162    /// </para>
163    /// <para></para>
164    /// </summary>
165    public virtual LinksConstants<TLinkAddress> Constants
166    {
167        [MethodImpl(MethodImplOptions.AggressiveInlining)]
168        get;
169    }
170
171    /// <summary>
172    /// <para>

```

```

171     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
172     /// </para>
173     /// </summary>
174     /// <param name="dataMemory">
175     /// <para>A data memory.</para>
176     /// </param>
177     /// <param name="indexMemory">
178     /// <para>A index memory.</para>
179     /// </param>
180     /// <param name="memoryReservationStep">
181     /// <para>A memory reservation step.</para>
182     /// </param>
183     /// <param name="constants">
184     /// <para>A constants.</para>
185     /// </param>
186     /// <param name="useLinkedList">
187     /// <para>A use linked list.</para>
188     /// </param>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
191     ↪ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants,
192     ↪ bool useLinkedList)
193     {
194         _dataMemory = dataMemory;
195         _indexMemory = indexMemory;
196         _dataMemoryReservationStepInBytes = memoryReservationStep * LinkDataPartSizeInBytes;
197         _indexMemoryReservationStepInBytes = memoryReservationStep *
198         ↪ LinkIndexPartSizeInBytes;
199         _useLinkedList = useLinkedList;
200         Constants = constants;
201     }
202
203     /// <summary>
204     /// <para>
205     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
206     /// </para>
207     /// </summary>
208     /// <param name="dataMemory">
209     /// <para>A data memory.</para>
210     /// </param>
211     /// <param name="indexMemory">
212     /// <para>A index memory.</para>
213     /// </param>
214     /// <param name="memoryReservationStep">
215     /// <para>A memory reservation step.</para>
216     /// </param>
217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
218     protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
219     ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
220     ↪ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
221     ↪ useLinkedList: true) { }
222
223     /// <summary>
224     /// <para>
225     /// Inits the data memory.
226     /// </para>
227     /// </summary>
228     /// <param name="dataMemory">
229     /// <para>The data memory.</para>
230     /// </param>
231     /// <param name="indexMemory">
232     /// <para>The index memory.</para>
233     /// </param>
234     [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 TLinkAddress Count(ICollection<TLinkAddress>? restriction)
319 {
320     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
321     if (restriction.Count == 0)
322     {
323         return Total;
324     }
325     var constants = Constants;
326     var any = constants.Any;
327     var index = restriction[constants.IndexPart];
328     if (restriction.Count == 1)
329     {
330         if (AreEqual(index, any))
331         {
332             return Total;
333         }
334         return Exists(index) ? GetOne() : GetZero();
335     }
336     if (restriction.Count == 2)
337     {
338         var value = restriction[1];
339         if (AreEqual(index, any))
340         {
341             if (AreEqual(value, any))
342             {
343                 return Total; // Any - как отсутствие ограничения
344             }
345             var externalReferencesRange = constants.ExternalReferencesRange;
346             if (externalReferencesRange.HasValue &&
347                 ⇨ externalReferencesRange.Value.Contains(value))
348             {
349                 return Add(ExternalSourcesTreeMethods.CountUsages(value),
350                     ⇨ ExternalTargetsTreeMethods.CountUsages(value));
351             }
352             else
353             {
354                 if (_useLinkedList)
355                 {
356                     return Add(InternalSourcesListMethods.CountUsages(value),
357                         ⇨ InternalTargetsTreeMethods.CountUsages(value));
358                 }
359                 else
360                 {
361                     return Add(InternalSourcesTreeMethods.CountUsages(value),
362                         ⇨ InternalTargetsTreeMethods.CountUsages(value));
363                 }
364             }
365         }
366         else
367         {
368             if (!Exists(index))
369             {
370                 return GetZero();
371             }
372             if (AreEqual(value, any))
373             {
374                 return GetOne();
375             }
376             ref var storedLinkValue = ref GetLinkDataPartReference(index);
377             if (AreEqual(storedLinkValue.Source, value) ||
378                 ⇨ AreEqual(storedLinkValue.Target, value))
379             {
380                 return GetOne();
381             }
382             return GetZero();
383         }
384     }
385     if (restriction.Count == 3)
386     {
387         var externalReferencesRange = constants.ExternalReferencesRange;
388         var source = restriction[constants.SourcePart];

```

```

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         TLinkAddress 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             else
449             {
450                 if (_useLinkedList ||
451                     ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
452                     ↪ InternalTargetsTreeMethods.CountUsages(target)))
453                 {
454                     link = InternalTargetsTreeMethods.Search(source, target);
455                 }
456                 else
457                 {
458                     link = InternalSourcesTreeMethods.Search(source, target);
459                 }
460             }
461         }
462     }
463 }

```



```

457         else
458         {
459             if (_useLinkedList ||
460                 ⇨ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
461                 ⇨ InternalTargetsTreeMethods.CountUsages(target)))
462             {
463                 link = InternalTargetsTreeMethods.Search(source, target);
464             }
465             else
466             {
467                 link = InternalSourcesTreeMethods.Search(source, target);
468             }
469             return AreEqual(link, constants.Null) ? GetZero() : GetOne();
470         }
471     }
472     else
473     {
474         if (!Exists(index))
475         {
476             return GetZero();
477         }
478         if (AreEqual(source, any) && AreEqual(target, any))
479         {
480             return GetOne();
481         }
482         ref var storedLinkValue = ref GetLinkDataPartReference(index);
483         if (!AreEqual(source, any) && !AreEqual(target, any))
484         {
485             if (AreEqual(storedLinkValue.Source, source) &&
486                 ⇨ AreEqual(storedLinkValue.Target, target))
487             {
488                 return GetOne();
489             }
490             return GetZero();
491         }
492         var value = default(TLinkAddress);
493         if (AreEqual(source, any))
494         {
495             value = target;
496         }
497         if (AreEqual(target, any))
498         {
499             value = source;
500         }
501         if (AreEqual(storedLinkValue.Source, value) ||
502             ⇨ AreEqual(storedLinkValue.Target, value))
503         {
504             return GetOne();
505         }
506         return GetZero();
507     }
508 }
509 throw new NotSupportedException("Другие размеры и способы ограничений не
510 ⇨ поддерживаются.");
511 }
512
513 /// <summary>
514 /// <para>
515 /// Eaches the handler.
516 /// </para>
517 /// <para></para>
518 /// </summary>
519 /// <param name="handler">
520 /// <para>The handler.</para>
521 /// <para></para>
522 /// </param>
523 /// <param name="restriction">
524 /// <para>The substitution.</para>
525 /// <para></para>
526 /// </param>
527 /// <exception cref="NotSupportedException">
528 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
529 /// <para></para>
530 /// </exception>
531 /// <returns>
532 /// <para>The link</para>
533 /// <para></para>

```

```

530 /// </returns>
531 [MethodImpl(MethodImplOptions.AggressiveInlining)]
532 public virtual TLinkAddress Each(IList<TLinkAddress>? restriction,
    ↳ ReadHandler<TLinkAddress>? handler)
533 {
534     var constants = Constants;
535     var @break = constants.Break;
536     if (restriction.Count == 0)
537     {
538         for (var link = GetOne(); LessOrEqualThan(link,
    ↳ GetHeaderReference().AllocatedLinks); link = Increment(link))
539         {
540             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
541             {
542                 return @break;
543             }
544         }
545         return @break;
546     }
547     var @continue = constants.Continue;
548     var any = constants.Any;
549     var index = restriction[constants.IndexPart];
550     if (restriction.Count == 1)
551     {
552         if (AreEqual(index, any))
553         {
554             return Each(Array.Empty<TLinkAddress>(), handler);
555         }
556         if (!Exists(index))
557         {
558             return @continue;
559         }
560         return handler(GetLinkStruct(index));
561     }
562     if (restriction.Count == 2)
563     {
564         var value = restriction[1];
565         if (AreEqual(index, any))
566         {
567             if (AreEqual(value, any))
568             {
569                 return Each(Array.Empty<TLinkAddress>(), handler);
570             }
571             if (AreEqual(Each(new Link<TLinkAddress>(index, value, any), handler),
    ↳ @break))
572             {
573                 return @break;
574             }
575             return Each(new Link<TLinkAddress>(index, any, value), handler);
576         }
577         else
578         {
579             if (!Exists(index))
580             {
581                 return @continue;
582             }
583             if (AreEqual(value, any))
584             {
585                 return handler(GetLinkStruct(index));
586             }
587             ref var storedLinkValue = ref GetLinkDataPartReference(index);
588             if (AreEqual(storedLinkValue.Source, value) ||
    AreEqual(storedLinkValue.Target, value))
589             {
590                 return handler(GetLinkStruct(index));
591             }
592             return @continue;
593         }
594     }
595 }
596 if (restriction.Count == 3)
597 {
598     var externalReferencesRange = constants.ExternalReferencesRange;
599     var source = restriction[constants.SourcePart];
600     var target = restriction[constants.TargetPart];
601     if (AreEqual(index, any))
602     {
603         if (AreEqual(source, any) && AreEqual(target, any))
604         {

```

```

605         return Each(Array.Empty<TLinkAddress>(), 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         TLinkAddress 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         else
678         {
679             if (_useLinkedList ||
680                 ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
681                 ↪ InternalTargetsTreeMethods.CountUsages(target)))
682             {
683                 link = InternalTargetsTreeMethods.Search(source, target);
684             }
685             else
686             {
687                 link = InternalSourcesTreeMethods.Search(source, target);
688             }
689         }
690     }
691     return link;
692 }

```

```

676         link = InternalTargetsTreeMethods.Search(source, target);
677     }
678     else
679     {
680         link = InternalSourcesTreeMethods.Search(source, target);
681     }
682 }
683 return AreEqual(link, constants.Null) ? @continue :
    ↪ 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(TLinkAddress);
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 TLinkAddress Update(IList<TLinkAddress>? restriction,
    ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
731 {
732     var constants = Constants;
733     var @null = constants.Null;
734     var externalReferencesRange = constants.ExternalReferencesRange;
735     var linkIndex = restriction[constants.IndexPart];
736     var before = GetLinkStruct(linkIndex);
737     ref var link = ref GetLinkDataPartReference(linkIndex);
738     var source = link.Source;
739     var target = link.Target;
740     ref var header = ref GetHeaderReference();
741     ref var rootAsSource = ref header.RootAsSource;
742     ref var rootAsTarget = ref header.RootAsTarget;
743     // Будет корректно работать только в том случае, если пространство выделенной связи
    ↪ предварительно заполнено нулями
744     if (!AreEqual(source, @null))
745     {
746         if (externalReferencesRange.HasValue &&
    ↪ externalReferencesRange.Value.Contains(source))
747     {

```

```

748         ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
749     }
750     else
751     {
752         if (_useLinkedList)
753         {
754             InternalSourcesListMethods.Detach(source, linkIndex);
755         }
756         else
757         {
758             InternalSourcesTreeMethods.Detach(ref
759                 ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
760         }
761     }
762     if (!AreEqual(target, @null))
763     {
764         if (externalReferencesRange.HasValue &&
765             ↪ externalReferencesRange.Value.Contains(target))
766         {
767             ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
768         }
769         else
770         {
771             InternalTargetsTreeMethods.Detach(ref
772                 ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
773         }
774     }
775     source = link.Source = substitution[constants.SourcePart];
776     target = link.Target = substitution[constants.TargetPart];
777     if (!AreEqual(source, @null))
778     {
779         if (externalReferencesRange.HasValue &&
780             ↪ externalReferencesRange.Value.Contains(source))
781         {
782             ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
783         }
784         else
785         {
786             if (_useLinkedList)
787             {
788                 InternalSourcesListMethods.AttachAsLast(source, linkIndex);
789             }
790             else
791             {
792                 InternalSourcesTreeMethods.Attach(ref
793                     ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
794             }
795         }
796     }
797     if (!AreEqual(target, @null))
798     {
799         if (externalReferencesRange.HasValue &&
800             ↪ externalReferencesRange.Value.Contains(target))
801         {
802             ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
803         }
804         else
805         {
806             InternalTargetsTreeMethods.Attach(ref
807                 ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
808         }
809     }
810     return handler?.Invoke(before, new Link<TLinkAddress>(linkIndex, source, target)) ??
811         ↪ Constants.Continue;
812 }
813
814 /// <remarks>
815 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
816 ↪ пространство
817 /// </remarks>
818 [MethodImpl(MethodImplOptions.AggressiveInlining)]
819 public virtual TLinkAddress Create(IList<TLinkAddress>? substitution,
820     ↪ WriteHandler<TLinkAddress>? handler)
821 {
822     ref var header = ref GetHeaderReference();
823     var freeLink = header.FirstFreeLink;
824     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
825             ↳ LinksLimitReachedException<TLinkAddress>(maximumPossibleInnerReference);
826     }
827     if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
828     {
829         _dataMemory.ReservedCapacity += _dataMemory.ReservationStepInBytes;
830         _indexMemory.ReservedCapacity += _indexMemory.ReservationStepInBytes;
831         SetPointers(_dataMemory, _indexMemory);
832         header = ref GetHeaderReference();
833         header.ReservedLinks = ConvertToAddress(_dataMemory.ReservedCapacity /
834             ↳ LinkDataPartSizeInBytes);
835     }
836     freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
837     _dataMemory.UsedCapacity += LinkDataPartSizeInBytes;
838     _indexMemory.UsedCapacity += LinkIndexPartSizeInBytes;
839 }
840 return handler?.Invoke(null, GetLinkStruct(freeLink)) ?? Constants.Continue;
841 }
842
843 /// <summary>
844 /// <para>
845 /// Deletes the substitution.
846 /// </para>
847 /// <para></para>
848 /// </summary>
849 /// <param name="restriction">
850 /// <para>The substitution.</para>
851 /// <para></para>
852 /// </param>
853 [MethodImpl(MethodImplOptions.AggressiveInlining)]
854 public virtual TLinkAddress Delete(ICollection<TLinkAddress>? restriction,
855     ↳ WriteHandler<TLinkAddress>? handler)
856 {
857     ref var header = ref GetHeaderReference();
858     var link = restriction[Constants.IndexPart];
859     var before = GetLinkStruct(link);
860     if (LessThan(link, header.AllocatedLinks))
861     {
862         UnusedLinksListMethods.AttachAsFirst(link);
863     }
864     else if (AreEqual(link, header.AllocatedLinks))
865     {
866         header.AllocatedLinks = Decrement(header.AllocatedLinks);
867         _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
868         _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
869         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
870         ↳ пока не дойдём до первой существующей связи
871         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
872         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
873             ↳ IsUnusedLink(header.AllocatedLinks))
874         {
875             UnusedLinksListMethods.Detach(header.AllocatedLinks);
876             header.AllocatedLinks = Decrement(header.AllocatedLinks);
877             _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
878             _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
879         }
880     }
881     return handler?.Invoke(before, null) ?? Constants.Continue;
882 }
883
884 /// <summary>
885 /// <para>
886 /// Gets the link struct using the specified link index.
887 /// </para>
888 /// <para></para>
889 /// </summary>
890 /// <param name="linkIndex">
891 /// <para>The link index.</para>
892 /// <para></para>
893 /// </param>

```

```

889 /// <returns>
890 /// <para>A list of t link</para>
891 /// <para></para>
892 /// </returns>
893 [MethodImpl(MethodImplOptions.AggressiveInlining)]
894 public IList<TLinkAddress>? GetLinkStruct(TLinkAddress linkIndex)
895 {
896     ref var link = ref GetLinkDataPartReference(linkIndex);
897     return new Link<TLinkAddress>(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<TLinkAddress> 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 /// </param>
951 /// <returns>
952 /// <para>A ref raw link data part of t link</para>
953 /// <para></para>
954 /// </returns>
955 [MethodImpl(MethodImplOptions.AggressiveInlining)]
956 protected abstract ref RawLinkDataPart<TLinkAddress>
957   → GetLinkDataPartReference(TLinkAddress linkIndex);
958
959 /// <summary>
960 /// <para>
961 /// Gets the link index part reference using the specified link index.
962 /// </para>
963 /// <para></para>
964 /// </summary>
965 /// <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<TLinkAddress>
    ↪ GetLinkIndexPartReference(TLinkAddress linkIndex);

973
974    /// <summary>
975    /// <para>
976    /// Determines whether this instance exists.
977    /// </para>
978    /// <para></para>
979    /// </summary>
980    /// <param name="link">
981    /// <para>The link.</para>
982    /// <para></para>
983    /// </param>
984    /// <returns>
985    /// <para>The bool</para>
986    /// <para></para>
987    /// </returns>
988    [MethodImpl(MethodImplOptions.AggressiveInlining)]
989    protected virtual bool Exists(TLinkAddress link)
990        => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
991            && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
992            && !IsUnusedLink(link);
993
994    /// <summary>
995    /// <para>
996    /// Determines whether this instance is unused link.
997    /// </para>
998    /// <para></para>
999    /// </summary>
1000    /// <param name="linkIndex">
1001    /// <para>The link index.</para>
1002    /// <para></para>
1003    /// </param>
1004    /// <returns>
1005    /// <para>The bool</para>
1006    /// <para></para>
1007    /// </returns>
1008    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1009    protected virtual bool IsUnusedLink(TLinkAddress linkIndex)
1010    {
1011        if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
1012            ↪ is not needed
1013        {
1014            // TODO: Reduce access to memory in different location (should be enough to use
1015            ↪ just linkIndexPart)
1016            ref var linkDataPart = ref GetLinkDataPartReference(linkIndex);
1017            ref var linkIndexPart = ref GetLinkIndexPartReference(linkIndex);
1018            return AreEqual(linkIndexPart.SizeAsTarget, default) &&
1019                ↪ !AreEqual(linkDataPart.Source, default);
1020        }
1021        else
1022        {
1023            return true;
1024        }
1025    }
1026
1027    /// <summary>
1028    /// <para>
1029    /// Gets the one.
1030    /// </para>
1031    /// <para></para>
1032    /// </summary>
1033    /// <returns>
1034    /// <para>The link</para>
1035    /// <para></para>
1036    /// </returns>
1037    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1038    protected virtual TLinkAddress GetOne() => _one;
1039
1040    /// <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 TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress second) =>
1092        ↪ _comparer.Compare(first, 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)]
    protected virtual bool LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
        ↪ _comparer.Compare(first, second) <= 0;

```

```

1113     /// <summary>
1114     /// <para>
1115     /// Determines whether this instance greater than.
1116     /// </para>
1117     /// <para></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(TLinkAddress first, TLinkAddress second) =>
1133         ↪ _comparer.Compare(first, second) > 0;
1134
1135     /// <summary>
1136     /// <para>
1137     /// Determines whether this instance greater or equal than.
1138     /// </para>
1139     /// <para></para>
1140     /// </summary>
1141     /// <param name="first">
1142     /// <para>The first.</para>
1143     /// <para></para>
1144     /// </param>
1145     /// <param name="second">
1146     /// <para>The second.</para>
1147     /// <para></para>
1148     /// </param>
1149     /// <returns>
1150     /// <para>The bool</para>
1151     /// <para></para>
1152     /// </returns>
1153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1154     protected virtual bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
1155         ↪ _comparer.Compare(first, second) >= 0;
1156
1157     /// <summary>
1158     /// <para>
1159     /// Converts the to int 64 using the specified value.
1160     /// </para>
1161     /// <para></para>
1162     /// </summary>
1163     /// <param name="value">
1164     /// <para>The value.</para>
1165     /// <para></para>
1166     /// </param>
1167     /// <returns>
1168     /// <para>The long</para>
1169     /// <para></para>
1170     /// </returns>
1171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1172     protected virtual long ConvertToInt64(TLinkAddress value) =>
1173         ↪ _addressToInt64Converter.Convert(value);
1174
1175     /// <summary>
1176     /// <para>
1177     /// Converts the to address using the specified value.
1178     /// </para>
1179     /// <para></para>
1180     /// </summary>
1181     /// <param name="value">
1182     /// <para>The value.</para>
1183     /// <para></para>
1184     /// </param>
1185     /// <returns>
1186     /// <para>The link</para>
1187     /// <para></para>
1188     /// </returns>
1189     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

1187     protected virtual TLinkAddress 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 TLinkAddress Add(TLinkAddress first, TLinkAddress second) =>
1209         ↪ Arithmetic<TLinkAddress>.Add(first, 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 TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
1230         ↪ Arithmetic<TLinkAddress>.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 TLinkAddress Increment(TLinkAddress link) =>
1247         ↪ Arithmetic<TLinkAddress>.Increment(link);
1248
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 TLinkAddress Decrement(TLinkAddress link) =>
        ↪ Arithmetic<TLinkAddress>.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{TLinkAddress}">
17     /// <seealso cref="ILinksListMethods{TLinkAddress}">
18     public unsafe class UnusedLinksListMethods<TLinkAddress> :
        ↪ AbsoluteCircularDoublyLinkedListMethods<TLinkAddress>, ILinksListMethods<TLinkAddress>
19     {
20         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
            ↪ = UncheckedConverter<TLinkAddress, long>.Default;
21         private readonly byte* _links;
22         private readonly byte* _header;
23
24         /// <summary>
25         /// <para>
26         /// Initializes a new <see cref="UnusedLinksListMethods"> instance.
27         /// </para>
28         /// <para></para>

```

```

29     /// </summary>
30     /// <param name="links">
31     /// <para>A links.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="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<TLinkAddress> GetHeaderReference() => ref
57     ↪ AsRef<LinksHeader<TLinkAddress>>(_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<TLinkAddress>
75     ↪ GetLinkDataPartReference(TLinkAddress link) => ref
76     ↪ AsRef<RawLinkDataPart<TLinkAddress>>(_links +
77     ↪ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
78     ↪ _addressToInt64Converter.Convert(link)));
79
80     /// <summary>
81     /// <para>
82     /// Gets the first.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLinkAddress GetFirst() => GetHeaderReference().FirstFreeLink;
92
93     /// <summary>
94     /// <para>
95     /// Gets the last.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <returns>
100    /// <para>The link</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected override TLinkAddress GetLast() => GetHeaderReference().LastFreeLink;
105
106    /// <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 TLinkAddress GetPrevious(TLinkAddress 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 TLinkAddress GetNext(TLinkAddress 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 TLinkAddress 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(TLinkAddress element) =>
162        ↪ GetHeaderReference().FirstFreeLink = 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(TLinkAddress element) =>
176        ↪ GetHeaderReference().LastFreeLink = element;

```

```

176     /// Sets the previous using the specified element.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <param name="element">
181     /// <para>The element.</para>
182     /// <para></para>
183     /// </param>
184     /// <param name="previous">
185     /// <para>The previous.</para>
186     /// <para></para>
187     /// </param>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override void SetPrevious(TLinkAddress element, TLinkAddress 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(TLinkAddress element, TLinkAddress 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(TLinkAddress size) => GetHeaderReference().FreeLinks =
222         ↪ size;
223 }
224 }

```

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<TLinkAddress> : IEquatable<RawLinkDataPart<TLinkAddress>>
17     {
18         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
19             ↪ EqualityComparer<TLinkAddress>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLinkDataPart<TLinkAddress>>.Size;

```

```

28     /// <summary>
29     /// <para>
30     /// The source.
31     /// </para>
32     /// <para></para>
33     /// </summary>
34     public TLinkAddress Source;
35     /// <summary>
36     /// <para>
37     /// The target.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public TLinkAddress 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<TLinkAddress> link ?
    ↪ Equals(link) : false;
59
60     /// <summary>
61     /// <para>
62     /// Determines whether this instance equals.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="other">
67     /// <para>The other.</para>
68     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The bool</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public bool Equals(RawLinkDataPart<TLinkAddress> other)
76     => _equalityComparer.Equals(Source, other.Source)
77     && _equalityComparer.Equals(Target, other.Target);
78
79     /// <summary>
80     /// <para>
81     /// Gets the hash code.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <returns>
86     /// <para>The int</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public override int GetHashCode() => (Source, Target).GetHashCode();
91
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public static bool operator ==(RawLinkDataPart<TLinkAddress> left,
    ↪ RawLinkDataPart<TLinkAddress> right) => left.Equals(right);
94
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public static bool operator !=(RawLinkDataPart<TLinkAddress> left,
    ↪ RawLinkDataPart<TLinkAddress> right) => !(left == right);
97 }
98 }

```

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<TLinkAddress> : IEquatable<RawLinkIndexPart<TLinkAddress>>
17     {
18         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
19             ↳ EqualityComparer<TLinkAddress>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLinkIndexPart<TLinkAddress>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The root as source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLinkAddress RootAsSource;
36
37         /// <summary>
38         /// <para>
39         /// The left as source.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLinkAddress LeftAsSource;
44
45         /// <summary>
46         /// <para>
47         /// The right as source.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public TLinkAddress RightAsSource;
52
53         /// <summary>
54         /// <para>
55         /// The size as source.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         public TLinkAddress SizeAsSource;
60
61         /// <summary>
62         /// <para>
63         /// The root as target.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         public TLinkAddress RootAsTarget;
68
69         /// <summary>
70         /// <para>
71         /// The left as target.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         public TLinkAddress LeftAsTarget;
76
77         /// <summary>
78         /// <para>
79         /// The right as target.
80         /// </para>
81         /// <para></para>
82         /// </summary>
83         public TLinkAddress RightAsTarget;
84
85         /// <summary>
86         /// <para>
87         /// The size as target.
88         /// </para>
89         /// <para></para>
90         /// </summary>
91         public TLinkAddress SizeAsTarget;
92     }
93 }

```

```

81     /// <para></para>
82     /// </summary>
83     public TLinkAddress SizeAsTarget;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equals.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="obj">
92     /// <para>The obj.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override bool Equals(object obj) => obj is RawLinkIndexPart<TLinkAddress> 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<TLinkAddress> 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<TLinkAddress> left,
        ↳ RawLinkIndexPart<TLinkAddress> right) => left.Equals(right);
142
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    public static bool operator !=(RawLinkIndexPart<TLinkAddress> left,
        ↳ RawLinkIndexPart<TLinkAddress> 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 TLinkAddress = 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{TLinkAddress}"/>
16    /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
17    public unsafe abstract class UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
18    ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>,
19    ↪ ILinksTreeMethods<TLinkAddress>
20    {
21        /// <summary>
22        /// <para>
23        /// The links data parts.
24        /// </para>
25        /// <para></para>
26        /// </summary>
27        protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
28        /// <summary>
29        /// <para>
30        /// The links index parts.
31        /// </para>
32        /// <para></para>
33        /// </summary>
34        protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
35        /// <summary>
36        /// <para>
37        /// The header.
38        /// </para>
39        /// <para></para>
40        /// </summary>
41        protected new readonly LinksHeader<TLinkAddress>* Header;
42
43        /// <summary>
44        /// <para>
45        /// Initializes a new <see
46        ↪ cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
47        /// </para>
48        /// <para></para>
49        /// </summary>
50        /// <param name="constants">
51        /// <para>A constants.</para>
52        /// <para></para>
53        /// </param>
54        /// <param name="linksDataParts">
55        /// <para>A links data parts.</para>
56        /// <para></para>
57        /// </param>
58        /// <param name="linksIndexParts">
59        /// <para>A links index parts.</para>
60        /// <para></para>
61        /// </param>
62        /// <param name="header">
63        /// <para>A header.</para>
64        /// <para></para>
65        /// </param>
66        [MethodImpl(MethodImplOptions.AggressiveInlining)]
67        protected UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
68        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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>

```

```

80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress value) => value == OUL; //
        ↪ 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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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 TLinkAddress Increment(TLinkAddress 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 TLinkAddress Decrement(TLinkAddress 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 TLinkAddress Add(TLinkAddress first, TLinkAddress 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 TLinkAddress Subtract(TLinkAddress first, TLinkAddress 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<TLinkAddress> 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<TLinkAddress>
    ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
380

```

```

381     /// <summary>
382     /// <para>
383     /// Gets the link index part reference using the specified link.
384     /// </para>
385     /// <para></para>
386     /// </summary>
387     /// <param name="link">
388     /// <para>The link.</para>
389     /// <para></para>
390     /// </param>
391     /// <returns>
392     /// <para>A ref raw link index part of t link</para>
393     /// <para></para>
394     /// </returns>
395     [MethodImpl(MethodImplOptions.AggressiveInlining)]
396     protected override ref RawLinkIndexPart<TLinkAddress>
397     ↪ GetLinkIndexPartReference(TLinkAddress link) => 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></para>
408     /// </param>
409     /// <param name="second">
410     /// <para>The second.</para>
411     /// <para></para>
412     /// </param>
413     /// <returns>
414     /// <para>The bool</para>
415     /// <para></para>
416     /// </returns>
417     [MethodImpl(MethodImplOptions.AggressiveInlining)]
418     protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
419     {
420         ref var firstLink = ref LinksDataParts[first];
421         ref var secondLink = ref LinksDataParts[second];
422         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
423             ↪ secondLink.Source, secondLink.Target);
424     }
425
426     /// <summary>
427     /// <para>
428     /// Determines whether this instance first is to the right of second.
429     /// </para>
430     /// <para></para>
431     /// </summary>
432     /// <param name="first">
433     /// <para>The first.</para>
434     /// <para></para>
435     /// </param>
436     /// <param name="second">
437     /// <para>The second.</para>
438     /// <para></para>
439     /// </param>
440     /// <returns>
441     /// <para>The bool</para>
442     /// <para></para>
443     /// </returns>
444     [MethodImpl(MethodImplOptions.AggressiveInlining)]
445     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
446     ↪ second)
447     {
448         ref var firstLink = ref LinksDataParts[first];
449         ref var secondLink = ref LinksDataParts[second];
450         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
451             ↪ secondLink.Source, secondLink.Target);
452     }
453 }

```

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 TLinkAddress = 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{TLinkAddress}"/>
16    /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
17    public unsafe abstract class UInt32ExternalLinksSizeBalancedTreeMethodsBase :
18    ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
19    {
20        /// <summary>
21        /// <para>
22        /// The links data parts.
23        /// </para>
24        /// <para></para>
25        /// </summary>
26        protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
27        /// <summary>
28        /// <para>
29        /// The links index parts.
30        /// </para>
31        /// <para></para>
32        /// </summary>
33        protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
34        /// <summary>
35        /// <para>
36        /// The header.
37        /// </para>
38        /// <para></para>
39        /// </summary>
40        protected new readonly LinksHeader<TLinkAddress>* Header;
41
42        /// <summary>
43        /// <para>
44        /// Initializes a new <see cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
45        ↪ instance.
46        /// </para>
47        /// <para></para>
48        /// </summary>
49        /// <param name="constants">
50        /// <para>A constants.</para>
51        /// <para></para>
52        /// </param>
53        /// <param name="linksDataParts">
54        /// <para>A links data parts.</para>
55        /// <para></para>
56        /// </param>
57        /// <param name="linksIndexParts">
58        /// <para>A links index parts.</para>
59        /// <para></para>
60        /// </param>
61        /// <param name="header">
62        /// <para>A header.</para>
63        /// <para></para>
64        /// </param>
65        [MethodImpl(MethodImplOptions.AggressiveInlining)]
66        protected UInt32ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
67        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
68        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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>

```

```

78     /// <returns>
79     /// <para>The link</para>
80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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 TLinkAddress Increment(TLinkAddress 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 TLinkAddress Decrement(TLinkAddress 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 TLinkAddress Add(TLinkAddress first, TLinkAddress 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 TLinkAddress Subtract(TLinkAddress first, TLinkAddress 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<TLinkAddress> 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<TLinkAddress>
    ↳ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
380
381 /// <summary>
382 /// <para>
383 /// Gets the link index part reference using the specified link.
384 /// </para>
385 /// <para></para>
386 /// </summary>
387 /// <param name="link">
388 /// <para>The link.</para>
389 /// <para></para>
390 /// </param>
391 /// <returns>
392 /// <para>A ref raw link index part of t link</para>
393 /// <para></para>
394 /// </returns>
395 [MethodImpl(MethodImplOptions.AggressiveInlining)]
396 protected override ref RawLinkIndexPart<TLinkAddress>
    ↳ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
397
398 /// <summary>
399 /// <para>
400 /// Determines whether this instance first is to the left of second.
401 /// </para>
402 /// <para></para>
403 /// </summary>
404 /// <param name="first">
405 /// <para>The first.</para>
406 /// <para></para>
407 /// </param>
408 /// <param name="second">
409 /// <para>The second.</para>
410 /// <para></para>
411 /// </param>
412 /// <returns>
413 /// <para>The bool</para>
414 /// <para></para>
415 /// </returns>
416 [MethodImpl(MethodImplOptions.AggressiveInlining)]
417 protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
418 {
419     ref var firstLink = ref LinksDataParts[first];
420     ref var secondLink = ref LinksDataParts[second];
421     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↳ secondLink.Source, secondLink.Target);
422 }
423
424 /// <summary>
425 /// <para>
426 /// Determines whether this instance first is to the right of second.
427 /// </para>
428 /// <para></para>
429 /// </summary>
430 /// <param name="first">
431 /// <para>The first.</para>
432 /// <para></para>
433 /// </param>
434 /// <param name="second">
435 /// <para>The second.</para>
436 /// <para></para>
437 /// </param>
438 /// <returns>
439 /// <para>The bool</para>
440 /// <para></para>
441 /// </returns>
442 [MethodImpl(MethodImplOptions.AggressiveInlining)]
443 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
    ↳ second)
444 {
445     ref var firstLink = ref LinksDataParts[first];
446     ref var secondLink = ref LinksDataParts[second];
447     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
    ↳ secondLink.Source, secondLink.Target);
448 }
449 }
450 }

```

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

```

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

```

```

72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLinkAddress GetRightReference(TLinkAddress 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     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLinkAddress GetLeft(TLinkAddress node) =>
92     ↪ 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    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLinkAddress GetRight(TLinkAddress node) =>
109    ↪ 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    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// </param>
123    /// </summary>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLinkAddress node, TLinkAddress 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    /// </param>
140    /// </summary>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
143    ↪ LinksIndexParts[node].RightAsSource = right;

```



```

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

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

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

```

1.53 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMeth

```

1 using System.Runtime.CompilerServices;
2 using TLinkAddress = System.UInt32;
3

```

```

4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 32 external links sources size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt32ExternalLinksSourcesSizeBalancedTreeMethods :
16    ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see cref="UInt32ExternalLinksSourcesSizeBalancedTreeMethods"/>
21        ↪ instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public UInt32ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
43        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
44        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
45        ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
46
47        /// <summary>
48        /// <para>
49        /// Gets the left reference using the specified node.
50        /// </para>
51        /// <para></para>
52        /// </summary>
53        /// <param name="node">
54        /// <para>The node.</para>
55        /// <para></para>
56        /// </param>
57        /// <returns>
58        /// <para>The ref link</para>
59        /// <para></para>
60        /// </returns>
61        [MethodImpl(MethodImplOptions.AggressiveInlining)]
62        protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
63        ↪ LinksIndexParts[node].LeftAsSource;
64
65        /// <summary>
66        /// <para>
67        /// Gets the right reference using the specified node.
68        /// </para>
69        /// <para></para>
70        /// </summary>
71        /// <param name="node">
72        /// <para>The node.</para>
73        /// <para></para>
74        /// </param>
75        /// <returns>
76        /// <para>The ref link</para>
77        /// <para></para>
78        /// </returns>
79        [MethodImpl(MethodImplOptions.AggressiveInlining)]
80        protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
81        ↪ LinksIndexParts[node].RightAsSource;
82    }
83 }

```

```

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 TLinkAddress GetLeft(TLinkAddress node) =>
92        ↪ 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 TLinkAddress GetRight(TLinkAddress node) =>
110        ↪ LinksIndexParts[node].RightAsSource;
111
112    /// <summary>
113    /// <para>
114    /// Sets the left using the specified node.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="node">
119    /// <para>The node.</para>
120    /// <para></para>
121    /// </param>
122    /// <param name="left">
123    /// <para>The left.</para>
124    /// <para></para>
125    /// </param>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
128        ↪ LinksIndexParts[node].LeftAsSource = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">
141    /// <para>The right.</para>
142    /// <para></para>
143    /// </param>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
146        ↪ LinksIndexParts[node].RightAsSource = right;
147
148    /// <summary>
149    /// <para>
150    /// Gets the size using the specified node.
151    /// </para>
152    /// <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 TLinkAddress GetSize(TLinkAddress node) =>
160         ↪ LinksIndexParts[node].SizeAsSource;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="node">
169     /// <para>The node.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="size">
173     /// <para>The size.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
178         ↪ LinksIndexParts[node].SizeAsSource = size;
179
180     /// <summary>
181     /// <para>
182     /// Gets the tree root.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <returns>
187     /// <para>The link</para>
188     /// <para></para>
189     /// </returns>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override TLinkAddress GetTreeRoot() => Header->RootAsSource;
192
193     /// <summary>
194     /// <para>
195     /// Gets the base part value using the specified node.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="node">
200     /// <para>The node.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The link</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
209         ↪ LinksDataParts[node].Source;
210
211     /// <summary>
212     /// <para>
213     /// Determines whether this instance first is to the left of second.
214     /// </para>
215     /// <para></para>
216     /// </summary>
217     /// <param name="firstSource">
218     /// <para>The first source.</para>
219     /// <para></para>
220     /// </param>
221     /// <param name="firstTarget">
222     /// <para>The first target.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="secondSource">
226     /// <para>The second source.</para>
227     /// <para></para>
228     /// </param>

```

```

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(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
236     => firstSource < secondSource || firstSource == secondSource && firstTarget <
    ↪ secondTarget;

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

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

```

1.54 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalance

```

1 using System.Runtime.CompilerServices;
2 using TLinkAddress = 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  {
17      /// <summary>
18      /// <para>
19      /// Initializes a new <see
20      ↪ cref="UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
42      ↪ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
43      ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
44      ↪ : base(constants, linksDataParts, 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress node) => ref
80      ↪ LinksIndexParts[node].RightAsTarget;
81
82      /// <summary>
83      /// <para>
84      /// Gets the left using the specified node.
85      /// </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 TLinkAddress GetLeft(TLinkAddress node) =>
92         ↪ LinksIndexParts[node].LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLinkAddress GetRight(TLinkAddress node) =>
110        ↪ 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>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
128        ↪ LinksIndexParts[node].LeftAsTarget = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">
141    /// <para>The right.</para>
142    /// <para></para>
143    /// </param>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
146        ↪ LinksIndexParts[node].RightAsTarget = right;
147
148    /// <summary>
149    /// <para>
150    /// Gets the size using the specified node.
151    /// </para>
152    /// <para></para>
153    /// </summary>
154    /// <param name="node">
155    /// <para>The node.</para>
156    /// <para></para>
157    /// </param>

```



```

154    /// <returns>
155    /// <para>The link</para>
156    /// <para></para>
157    /// </returns>
158    [MethodImpl(MethodImplOptions.AggressiveInlining)]
159    protected override TLinkAddress GetSize(TLinkAddress node) =>
160        ↳ 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetTreeRoot() => Header->RootAsTarget;
192
193    /// <summary>
194    /// <para>
195    /// Gets the base part value using the specified node.
196    /// </para>
197    /// <para></para>
198    /// </summary>
199    /// <param name="node">
200    /// <para>The node.</para>
201    /// <para></para>
202    /// </param>
203    /// <returns>
204    /// <para>The link</para>
205    /// <para></para>
206    /// </returns>
207    [MethodImpl(MethodImplOptions.AggressiveInlining)]
208    protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
209        ↳ LinksDataParts[node].Target;
210
211    /// <summary>
212    /// <para>
213    /// Determines whether this instance first is to the left of second.
214    /// </para>
215    /// <para></para>
216    /// </summary>
217    /// <param name="firstSource">
218    /// <para>The first source.</para>
219    /// <para></para>
220    /// </param>
221    /// <param name="firstTarget">
222    /// <para>The first target.</para>
223    /// <para></para>
224    /// </param>
225    /// <param name="secondSource">
226    /// <para>The second source.</para>
227    /// <para></para>
228    /// </param>
229    /// <param name="secondTarget">
230    /// <para>The second target.</para>
231    /// <para></para>
232    /// </param>

```

```

229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
236     => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
    ↪ secondSource;

237
238     /// <summary>
239     /// <para>
240     /// Determines whether this instance first is to the right of second.
241     /// </para>
242     /// <para></para>
243     /// </summary>
244     /// <param name="firstSource">
245     /// <para>The first source.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="firstTarget">
249     /// <para>The first target.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="secondSource">
253     /// <para>The second source.</para>
254     /// <para></para>
255     /// </param>
256     /// <param name="secondTarget">
257     /// <para>The second target.</para>
258     /// <para></para>
259     /// </param>
260     /// <returns>
261     /// <para>The bool</para>
262     /// <para></para>
263     /// </returns>
264     [MethodImpl(MethodImplOptions.AggressiveInlining)]
265     protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress node)
280     {
281         ref var link = ref LinksIndexParts[node];
282         link.LeftAsTarget = Zero;
283         link.RightAsTarget = Zero;
284         link.SizeAsTarget = Zero;
285     }
286 }
287 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using TLinkAddress = 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 :
    ↳ UInt32ExternalLinksSizeBalancedTreeMethodsBase
16  {
17      /// <summary>
18      /// <para>
19      /// Initializes a new <see cref="UInt32ExternalLinksTargetsSizeBalancedTreeMethods"/>
    ↳ 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 UInt32ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
    ↳ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress node) =>
92         ↪ LinksIndexParts[node].LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLinkAddress GetRight(TLinkAddress node) =>
110        ↪ 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>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
128        ↪ LinksIndexParts[node].LeftAsTarget = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">
141    /// <para>The right.</para>
142    /// <para></para>
143    /// </param>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
146        ↪ LinksIndexParts[node].RightAsTarget = right;
147
148    /// <summary>
149    /// <para>
150    /// Gets the size using the specified node.
151    /// </para>
152    /// <para></para>
153    /// </summary>
154    /// <param name="node">
155    /// <para>The node.</para>
156    /// <para></para>
157    /// </param>
158    /// <returns>
159    /// <para>The link</para>
160    /// <para></para>
161    /// </returns>
162    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

159     protected override TLinkAddress GetSize(TLinkAddress node) =>
160         ↪ 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetTreeRoot() => Header->RootAsTarget;
192
193     /// <summary>
194     /// <para>
195     /// Gets the base part value using the specified node.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="node">
200     /// <para>The node.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The link</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
209         ↪ LinksDataParts[node].Target;
210
211     /// <summary>
212     /// <para>
213     /// Determines whether this instance first is to the left of second.
214     /// </para>
215     /// <para></para>
216     /// </summary>
217     /// <param name="firstSource">
218     /// <para>The first source.</para>
219     /// <para></para>
220     /// </param>
221     /// <param name="firstTarget">
222     /// <para>The first target.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="secondSource">
226     /// <para>The second source.</para>
227     /// <para></para>
228     /// </param>
229     /// <param name="secondTarget">
230     /// <para>The second target.</para>
231     /// <para></para>
232     /// </param>
233     /// <returns>
234     /// <para>The bool</para>
235     /// <para></para>
236     /// </returns>

```

```

234 [MethodImpl(MethodImplOptions.AggressiveInlining)]
235 protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
236 => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
    ↪ secondSource;

237
238 /// <summary>
239 /// <para>
240 /// Determines whether this instance first is to the right of second.
241 /// </para>
242 /// <para></para>
243 /// </summary>
244 /// <param name="firstSource">
245 /// <para>The first source.</para>
246 /// <para></para>
247 /// </param>
248 /// <param name="firstTarget">
249 /// <para>The first target.</para>
250 /// <para></para>
251 /// </param>
252 /// <param name="secondSource">
253 /// <para>The second source.</para>
254 /// <para></para>
255 /// </param>
256 /// <param name="secondTarget">
257 /// <para>The second target.</para>
258 /// <para></para>
259 /// </param>
260 /// <returns>
261 /// <para>The bool</para>
262 /// <para></para>
263 /// </returns>
264 [MethodImpl(MethodImplOptions.AggressiveInlining)]
265 protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress 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/UInt32InternalLinksRecursionlessSizeBalancedTreeM

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLinkAddress = 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{TLinkAddress}"/>
16    public unsafe abstract class UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
17    {

```

```

18     /// <summary>
19     /// <para>
20     /// The links data parts.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
25     /// <summary>
26     /// <para>
27     /// The links index parts.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
32     /// <summary>
33     /// <para>
34     /// The header.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     protected new readonly LinksHeader<TLinkAddress>* Header;
39
40     /// <summary>
41     /// <para>
42     /// Initializes a new <see
43     ↪ cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="constants">
48     /// <para>A constants.</para>
49     /// </param>
50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// </param>
53     /// <param name="linksIndexParts">
54     /// <para>A links index parts.</para>
55     /// </param>
56     /// <param name="header">
57     /// <para>A header.</para>
58     /// </param>
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     protected UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
61     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
62     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
63     {
64         LinksDataParts = linksDataParts;
65         LinksIndexParts = linksIndexParts;
66         Header = header;
67     }
68
69     /// <summary>
70     /// <para>
71     /// Gets the zero.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <returns>
76     /// <para>The link</para>
77     /// <para></para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     protected override TLinkAddress GetZero() => 0U;
81
82     /// <summary>
83     /// <para>
84     /// Determines whether this instance equal to zero.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="value">
89     /// <para>The value.</para>
90     /// </param>
91     /// <para></para>
92     /// </summary>

```

```

93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override bool EqualToZero(TLinkAddress value) => value == 0U;
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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress second) =>
180         ↪ 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(TLinkAddress value) => true; // value >=
198         ↪ 0 is always true for ulong
199
200     /// <summary>
201     /// <para>
202     /// Determines whether this instance less or equal than zero.
203     /// </para>
204     /// <para></para>
205     /// </summary>
206     /// <param name="value">
207     /// <para>The value.</para>
208     /// <para></para>
209     /// </param>
210     /// <returns>
211     /// <para>The bool</para>
212     /// <para></para>
213     /// </returns>
214     [MethodImpl(MethodImplOptions.AggressiveInlining)]
215     protected override bool LessOrEqualThanZero(TLinkAddress value) => value == OUL; //
216         ↪ value is always >= 0 for ulong
217
218     /// <summary>
219     /// <para>
220     /// Determines whether this instance less or equal than.
221     /// </para>
222     /// <para></para>
223     /// </summary>
224     /// <param name="first">
225     /// <para>The first.</para>
226     /// <para></para>
227     /// </param>
228     /// <param name="second">
229     /// <para>The second.</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 LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
238         ↪ first <= second;
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance less than zero.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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 TLinkAddress Increment(TLinkAddress 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 TLinkAddress Decrement(TLinkAddress 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 TLinkAddress Add(TLinkAddress first, TLinkAddress 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 TLinkAddress Subtract(TLinkAddress first, TLinkAddress 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<TLinkAddress>
    ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
366
367    /// <summary>
368    /// <para>
369    /// Gets the link index part reference using the specified link.
370    /// </para>
371    /// <para></para>
372    /// </summary>
373    /// <param name="link">
374    /// <para>The link.</para>
375    /// <para></para>
376    /// </param>
377    /// <returns>
378    /// <para>A ref raw link index part of t link</para>
379    /// <para></para>
380    /// </returns>
381    [MethodImpl(MethodImplOptions.AggressiveInlining)]
382    protected override ref RawLinkIndexPart<TLinkAddress>
    ↪ GetLinkIndexPartReference(TLinkAddress 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 FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress
    ↪ 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(TLinkAddress first, TLinkAddress
    ↪ second) => GetKeyPartValue(first) > GetKeyPartValue(second);
425 }
426 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLinkAddress = 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{TLinkAddress}"/>
16    public unsafe abstract class UInt32InternalLinksSizeBalancedTreeMethodsBase :
    ↪ InternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>
17    {
18        /// <summary>
19        /// <para>
20        /// The links data parts.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
25        /// <summary>
26        /// <para>
27        /// The links index parts.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
32        /// <summary>
33        /// <para>
34        /// The header.
35        /// </para>
36        /// <para></para>
37        /// </summary>
38        protected new readonly LinksHeader<TLinkAddress>* Header;
39

```

```

40    /// <summary>
41    /// <para>
42    /// Initializes a new <see cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
    ↪ 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    /// <param name="header">
59    /// <para>A header.</para>
60    /// <para></para>
61    /// </param>
62    [MethodImpl(MethodImplOptions.AggressiveInlining)]
63    protected UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
    ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64    {
65        LinksDataParts = linksDataParts;
66        LinksIndexParts = linksIndexParts;
67        Header = header;
68    }
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 TLinkAddress GetZero() => 0U;
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(TLinkAddress value) => value == 0U;
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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress value) => value == OUL; //
    ↪ value is always >= 0 for ulong
214
215    /// <summary>
216    /// <para>
217    /// Determines whether this instance less or equal than.
218    /// </para>
219    /// <para></para>
220    /// </summary>
221    /// <param name="first">
222    /// <para>The first.</para>
223    /// <para></para>
224    /// </param>
225    /// <param name="second">
226    /// <para>The second.</para>
227    /// <para></para>
228    /// </param>
229    /// <returns>
230    /// <para>The bool</para>
231    /// <para></para>
232    /// </returns>
233    [MethodImpl(MethodImplOptions.AggressiveInlining)]
234    protected override bool LessOrEqualThan(TLinkAddress first, TLinkAddress 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(TLinkAddress 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(TLinkAddress first, TLinkAddress 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 TLinkAddress Increment(TLinkAddress 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 TLinkAddress Decrement(TLinkAddress 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 TLinkAddress Add(TLinkAddress first, TLinkAddress 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 TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
349        ↪ first - second;
350
351    /// <summary>
352    /// <para>
353    /// Gets the link data part reference using the specified link.
354    /// </para>
355    /// <para></para>
356    /// </summary>
357    /// <param name="link">
358    /// <para>The link.</para>
359    /// <para></para>
360    /// </param>
361    /// <returns>
362    /// <para>A ref raw link data part of t link</para>
363    /// <para></para>
364    /// </returns>
365    [MethodImpl(MethodImplOptions.AggressiveInlining)]
366    protected override ref RawLinkDataPart<TLinkAddress>
367        ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
368
369    /// <summary>
370    /// <para>
371    /// Gets the link index part reference using the specified link.
372    /// </para>
373    /// <para></para>
374    /// </summary>
375    /// <param name="link">
376    /// <para>The link.</para>
377    /// <para></para>
378    /// </param>
379    /// <returns>
380    /// <para>A ref raw link index part of t link</para>
381    /// <para></para>
382    /// </returns>
383    [MethodImpl(MethodImplOptions.AggressiveInlining)]
384    protected override ref RawLinkIndexPart<TLinkAddress>
385        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
386
387    /// <summary>
388    /// <para>
389    /// Determines whether this instance first is to the left of second.
390    /// </para>
391    /// <para></para>
392    /// </summary>
393    /// <param name="first">
394    /// <para>The first.</para>
395    /// <para></para>
396    /// </param>
397    /// <param name="second">
398    /// <para>The second.</para>
399    /// <para></para>
400    /// </param>
401    /// <returns>
402    /// <para>The bool</para>
403    /// <para></para>
404    /// </returns>
405    [MethodImpl(MethodImplOptions.AggressiveInlining)]
406    protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress
407        ↪ second) => GetKeyPartValue(first) < GetKeyPartValue(second);
408
409    /// <summary>
410    /// <para>
411    /// Determines whether this instance first is to the right of second.
412    /// </para>
413    /// <para></para>
414    /// </summary>
415    /// <param name="first">
416    /// <para>The first.</para>
417    /// <para></para>
418    /// </param>
419    /// <param name="second">
420    /// <para>The second.</para>
421    /// <para></para>
422    /// </param>
423    /// <returns>
424    /// <para>The bool</para>
425    /// <para></para>
426    /// </returns>
427    [MethodImpl(MethodImplOptions.AggressiveInlining)]
428    protected override bool FirstIsToRightOfSecond(TLinkAddress first, TLinkAddress
429        ↪ second) => GetKeyPartValue(first) > GetKeyPartValue(second);

```

```

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(TLinkAddress first, TLinkAddress
        ↪ second) => GetKeyPartValue(first) > GetKeyPartValue(second);
425 }
426 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Generic
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources linked list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLinkAddress}"/>
15     public unsafe class UInt32InternalLinksSourcesLinkedListMethods :
        ↪ InternalLinksSourcesLinkedListMethods<TLinkAddress>
16     {
17         private readonly RawLinkDataPart<TLinkAddress>* _linksDataParts;
18         private readonly RawLinkIndexPart<TLinkAddress>* _linksIndexParts;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt32InternalLinksSourcesLinkedListMethods"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="constants">
27         /// <para>A constants.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="linksDataParts">
31         /// <para>A links data parts.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="linksIndexParts">
35         /// <para>A links index parts.</para>
36         /// <para></para>
37         /// </param>
38         [MethodImpl(MethodImplOptions.AggressiveInlining)]
39         public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress>
            ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
            ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
40         {
41             _linksDataParts = linksDataParts;
42             _linksIndexParts = linksIndexParts;
43         }
44
45         /// <summary>
46         /// <para>
47         /// Gets the link data part reference using the specified link.
48         /// </para>
49         /// <para></para>
50         /// <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<TLinkAddress>
        ↪ GetLinkDataPartReference(TLinkAddress 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<TLinkAddress>
        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref _linksIndexParts[link];
79 }
80 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
        ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress left) =>
    ↳ LinksIndexParts[node].LeftAsSource = left;

126
127 /// <summary>
128 /// <para>
129 /// Sets the right using the specified node.
130 /// </para>
131 /// <para></para>
132 /// </summary>
133 /// <param name="node">
134 /// <para>The node.</para>
135 /// <para></para>
136 /// </param>
137 /// <param name="right">
138 /// <para>The right.</para>
139 /// <para></para>
140 /// </param>
141 [MethodImpl(MethodImplOptions.AggressiveInlining)]
142 protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
    ↳ LinksIndexParts[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 TLinkAddress GetSize(TLinkAddress 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(TLinkAddress node, TLinkAddress size) =>
    ↳ LinksIndexParts[node].SizeAsSource = 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 TLinkAddress GetTreeRoot(TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress node) =>
211         ↪ 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 TLinkAddress GetKeyPartValue(TLinkAddress node) =>
229         ↪ LinksDataParts[node].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(TLinkAddress node)
243     {
244         ref var link = ref LinksIndexParts[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>
260     /// <param name="target">
261     /// <para>The target.</para>
262     /// <para></para>
263     /// </param>
264     /// <returns>
265     /// <para>The link</para>
266     /// <para></para>
267     /// </returns>
268     public override TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
269         ↪ SearchCore(GetTreeRoot(source), target);
270 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = 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 :
16         ↳ UInt32InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32InternalLinksSourcesSizeBalancedTreeMethods"/>
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 UInt32InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
43             ↳ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
44             ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
45             ↳ : base(constants, linksDataParts, linksIndexParts, header) { }
46
47         /// <summary>
48         /// <para>
49         /// Gets the left reference using the specified node.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="node">
54         /// <para>The node.</para>
55         /// <para></para>
56         /// </param>
57         /// <returns>
58         /// <para>The ref link</para>
59         /// <para></para>
60         /// </returns>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
63             ↳ LinksIndexParts[node].LeftAsSource;
64
65         /// <summary>
66         /// <para>
67         /// Gets the right reference using the specified node.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         /// <param name="node">
72         /// <para>The node.</para>
73         /// <para></para>
74         /// </param>
75         /// <returns>
76         /// <para>The ref link</para>
77         /// <para></para>
78         /// </returns>

```

```

72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLinkAddress GetRightReference(TLinkAddress 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     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLinkAddress GetLeft(TLinkAddress node) =>
92     ↪ 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    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLinkAddress GetRight(TLinkAddress node) =>
109    ↪ 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    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// </param>
123    /// </summary>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLinkAddress node, TLinkAddress 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    /// </param>
140    /// </summary>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
143    ↪ LinksIndexParts[node].RightAsSource = right;

```



```

144    /// <summary>
145    /// <para>
146    /// Gets the size using the specified node.
147    /// </para>
148    /// <para></para>
149    /// </summary>
150    /// <param name="node">
151    /// <para>The node.</para>
152    /// <para></para>
153    /// </param>
154    /// <returns>
155    /// <para>The link</para>
156    /// <para></para>
157    /// </returns>
158    [MethodImpl(MethodImplOptions.AggressiveInlining)]
159    protected override TLinkAddress GetSize(TLinkAddress node) =>
160        ↪ LinksIndexParts[node].SizeAsSource;
161
162    /// <summary>
163    /// <para>
164    /// Sets the size using the specified node.
165    /// </para>
166    /// <para></para>
167    /// </summary>
168    /// <param name="node">
169    /// <para>The node.</para>
170    /// <para></para>
171    /// </param>
172    /// <param name="size">
173    /// <para>The size.</para>
174    /// <para></para>
175    /// </param>
176    [MethodImpl(MethodImplOptions.AggressiveInlining)]
177    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
178        ↪ LinksIndexParts[node].SizeAsSource = size;
179
180    /// <summary>
181    /// <para>
182    /// Gets the tree root using the specified node.
183    /// </para>
184    /// <para></para>
185    /// </summary>
186    /// <param name="node">
187    /// <para>The node.</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 TLinkAddress GetTreeRoot(TLinkAddress node) =>
196        ↪ LinksIndexParts[node].RootAsSource;
197
198    /// <summary>
199    /// <para>
200    /// Gets the base part value using the specified node.
201    /// </para>
202    /// <para></para>
203    /// </summary>
204    /// <param name="node">
205    /// <para>The node.</para>
206    /// <para></para>
207    /// </param>
208    /// <returns>
209    /// <para>The link</para>
210    /// <para></para>
211    /// </returns>
212    [MethodImpl(MethodImplOptions.AggressiveInlining)]
213    protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
214        ↪ LinksDataParts[node].Source;
215
216    /// <summary>
217    /// <para>
218    /// Gets the key part value using the specified node.
219    /// </para>
220    /// <para></para>
221    /// </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 TLinkAddress GetKeyPartValue(TLinkAddress node) =>
228         ↪ 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(TLinkAddress node)
242     {
243         ref var link = ref LinksIndexParts[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 TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
268         ↪ SearchCore(GetTreeRoot(source), target);

```

1.61 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = 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>

```

```

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 UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress node) =>
109        ↪ LinksIndexParts[node].RightAsTarget;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="node">
118    /// <para>The node.</para>
119    /// <para></para>
120    /// </param>
121    /// <param name="left">
122    /// <para>The left.</para>
123    /// <para></para>
124    /// </param>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
127        ↪ LinksIndexParts[node].LeftAsTarget = left;
128
129    /// <summary>
130    /// <para>
131    /// Sets the right using the specified node.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="node">
136    /// <para>The node.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="right">
140    /// <para>The right.</para>
141    /// <para></para>
142    /// </param>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
145        ↪ LinksIndexParts[node].RightAsTarget = 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 TLinkAddress GetSize(TLinkAddress node) =>
163        ↪ LinksIndexParts[node].SizeAsTarget;
164
165    /// <summary>
166    /// <para>
167    /// Sets the size using the specified node.
168    /// </para>
169    /// <para></para>
170    /// </summary>
171    /// <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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetTreeRoot(TLinkAddress node) =>
195        ↪ LinksIndexParts[node].RootAsTarget;
196
197    /// <summary>
198    /// <para>
199    /// Gets the base part value using the specified node.
200    /// </para>
201    /// <para></para>
202    /// </summary>
203    /// <param name="node">
204    /// <para>The node.</para>
205    /// <para></para>
206    /// </param>
207    /// <returns>
208    /// <para>The link</para>
209    /// <para></para>
210    /// </returns>
211    [MethodImpl(MethodImplOptions.AggressiveInlining)]
212    protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
213        ↪ LinksDataParts[node].Target;
214
215    /// <summary>
216    /// <para>
217    /// Gets the key part value using the specified node.
218    /// </para>
219    /// <para></para>
220    /// </summary>
221    /// <param name="node">
222    /// <para>The node.</para>
223    /// <para></para>
224    /// </param>
225    /// <returns>
226    /// <para>The link</para>
227    /// <para></para>
228    /// </returns>
229    [MethodImpl(MethodImplOptions.AggressiveInlining)]
230    protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
231        ↪ LinksDataParts[node].Source;
232
233    /// <summary>
234    /// <para>
235    /// Clears the node using the specified node.
236    /// </para>
237    /// <para></para>
238    /// </summary>
239    /// <param name="node">
240    /// <para>The node.</para>
241    /// <para></para>
242    /// </param>
243    [MethodImpl(MethodImplOptions.AggressiveInlining)]
244    protected override void ClearNode(TLinkAddress node)
245    {

```

```

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 TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
        ↪ SearchCore(GetTreeRoot(target), source);
267 }
268 }

```

1.62 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMetho

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksTargetsSizeBalancedTreeMethods :
        ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32InternalLinksTargetsSizeBalancedTreeMethods"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <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 UInt32InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetSize(TLinkAddress node) =>
162         ↪ LinksIndexParts[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)]
179     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
180         ↪ LinksIndexParts[node].SizeAsTarget = size;
181
182     /// <summary>
183     /// <para>
184     /// Gets the tree root using the specified node.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="node">
189     /// <para>The node.</para>
190     /// <para></para>
191     /// </param>
192     /// <returns>
193     /// <para>The link</para>
194     /// <para></para>
195     /// </returns>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

193     protected override TLinkAddress GetTreeRoot(TLinkAddress node) =>
194         ↳ 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 TLinkAddress GetBasePartValue(TLinkAddress node) =>
212         ↳ LinksDataParts[node].Target;
213
214     /// <summary>
215     /// <para>
216     /// Gets the key part value using the specified node.
217     /// </para>
218     /// <para></para>
219     /// </summary>
220     /// <param name="node">
221     /// <para>The node.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The link</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
230         ↳ LinksDataParts[node].Source;
231
232     /// <summary>
233     /// <para>
234     /// Clears the node using the specified node.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="node">
239     /// <para>The node.</para>
240     /// <para></para>
241     /// </param>
242     [MethodImpl(MethodImplOptions.AggressiveInlining)]
243     protected override void ClearNode(TLinkAddress node)
244     {
245         ref var link = ref LinksIndexParts[node];
246         link.LeftAsTarget = Zero;
247         link.RightAsTarget = Zero;
248         link.SizeAsTarget = Zero;
249     }
250
251     /// <summary>
252     /// <para>
253     /// Searches the source.
254     /// </para>
255     /// <para></para>
256     /// </summary>
257     /// <param name="source">
258     /// <para>The source.</para>
259     /// <para></para>
260     /// </param>
261     /// <param name="target">
262     /// <para>The target.</para>
263     /// <para></para>
264     /// </param>
265     /// <returns>
266     /// <para>The link</para>
267     /// <para></para>
268     /// </returns>
269     public override TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
270         ↳ 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 TLinkAddress = 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{TLinkAddress}"/>
19     public unsafe class UInt32SplitMemoryLinks : SplitMemoryLinksBase<TLinkAddress>
20     {
21         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalSourceTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalTargetTreeMethods;
24         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalTargetTreeMethods;
25         private LinksHeader<TLinkAddress>* _header;
26         private RawLinkDataPart<TLinkAddress>* _linksDataParts;
27         private RawLinkIndexPart<TLinkAddress>* _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
45             ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
46
47         /// <summary>
48         /// <para>
49         /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> 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 UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
67             ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
68             ↪ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
69             ↪ IndexTreeType.Default, useLinkedList: true) { }
70
71         /// <summary>
72         /// <para>
73         /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
74         /// </para>
75         /// <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<TLinkAddress> 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<TLinkAddress> 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
                ↳ UInt32InternalLinksTargetsSizeBalancedTreeMethods(Constants,
                ↳ _linksDataParts, _linksIndexParts, _header);
130            _createExternalTargetTreeMethods = () => new
                ↳ UInt32ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
                ↳ _linksDataParts, _linksIndexParts, _header);
131        }
132        else
133        {

```

```

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

```

```

201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override ref LinksHeader<TLinkAddress> 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<TLinkAddress>
220     ↪ GetLinkDataPartReference(TLinkAddress linkIndex) => 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<TLinkAddress>
238     ↪ GetLinkIndexPartReference(TLinkAddress 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(TLinkAddress first, TLinkAddress second) => first ==
260     ↪ second;
261
262 /// <summary>
263 /// <para>
264 /// Determines whether this instance less than.
265 /// </para>
266 /// <para></para>
267 /// </summary>
268 /// <param name="first">
269 /// <para>The first.</para>
270 /// <para></para>
271 /// </param>
272 /// <param name="second">
273 /// <para>The second.</para>
274 /// <para></para>
275 /// </param>
276 /// <returns>
277 /// <para>The bool</para>
278 /// <para></para>
279 /// </returns>

```

```

276    /// </returns>
277    [MethodImpl(MethodImplOptions.AggressiveInlining)]
278    protected override bool LessThan(TLinkAddress first, TLinkAddress second) => first <
    ↪     second;

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

300    /// <summary>
301    /// <para>
302    /// Determines whether this instance greater than.
303    /// </para>
304    /// <para></para>
305    /// </summary>
306    /// <param name="first">
307    /// <para>The first.</para>
308    /// <para></para>
309    /// </param>
310    /// <param name="second">
311    /// <para>The second.</para>
312    /// <para></para>
313    /// </param>
314    /// <returns>
315    /// <para>The bool</para>
316    /// <para></para>
317    /// </returns>
318    [MethodImpl(MethodImplOptions.AggressiveInlining)]
319    protected override bool GreaterThan(TLinkAddress first, TLinkAddress second) => first >
    ↪     second;

321    /// <summary>
322    /// <para>
323    /// Determines whether this instance greater or equal than.
324    /// </para>
325    /// <para></para>
326    /// </summary>
327    /// <param name="first">
328    /// <para>The first.</para>
329    /// <para></para>
330    /// </param>
331    /// <param name="second">
332    /// <para>The second.</para>
333    /// <para></para>
334    /// </param>
335    /// <returns>
336    /// <para>The bool</para>
337    /// <para></para>
338    /// </returns>
339    [MethodImpl(MethodImplOptions.AggressiveInlining)]
340    protected override bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
    ↪     first >= second;

342    /// <summary>
343    /// <para>
344    /// Gets the zero.
345    /// </para>
346    /// <para></para>
347    /// </summary>
348    /// <returns>

```

```

350    /// <para>The link</para>
351    /// <para></para>
352    /// </returns>
353    [MethodImpl(MethodImplOptions.AggressiveInlining)]
354    protected override TLinkAddress 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 TLinkAddress 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(TLinkAddress 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 TLinkAddress ConvertToAddress(long value) => (TLinkAddress)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 TLinkAddress Add(TLinkAddress first, TLinkAddress 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 TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
444         ↪ first - second;
445
446     /// <summary>
447     /// <para>
448     /// Increments the link.
449     /// </para>
450     /// <para></para>
451     /// </summary>
452     /// <param name="link">
453     /// <para>The link.</para>
454     /// <para></para>
455     /// </param>
456     /// <returns>
457     /// <para>The link</para>
458     /// <para></para>
459     /// </returns>
460     [MethodImpl(MethodImplOptions.AggressiveInlining)]
461     protected override TLinkAddress Increment(TLinkAddress link) => ++link;
462
463     /// <summary>
464     /// <para>
465     /// Decrements the link.
466     /// </para>
467     /// <para></para>
468     /// </summary>
469     /// <param name="link">
470     /// <para>The link.</para>
471     /// <para></para>
472     /// </param>
473     /// <returns>
474     /// <para>The link</para>
475     /// <para></para>
476     /// </returns>
477     [MethodImpl(MethodImplOptions.AggressiveInlining)]
478     protected override TLinkAddress Decrement(TLinkAddress link) => --link;
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 TLinkAddress = 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{TLinkAddress}"/>
16     public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<TLinkAddress>
17     {
18         private readonly RawLinkDataPart<TLinkAddress>* _links;
19         private readonly LinksHeader<TLinkAddress>* _header;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.

```



```

24     /// </para>
25     /// <para></para>
26     /// </summary>
27     /// <param name="links">
28     /// <para>A links.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="header">
32     /// <para>A header.</para>
33     /// <para></para>
34     /// </param>
35     [MethodImpl(MethodImplOptions.AggressiveInlining)]
36     public UInt32UnusedLinksListMethods(RawLinkDataPart<TLinkAddress>* links,
    → LinksHeader<TLinkAddress>* header)
    : base((byte*)links, (byte*)header)
37     {
38     {
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<TLinkAddress>
    → GetLinkDataPartReference(TLinkAddress link) => ref _links[link];
59
60     /// <summary>
61     /// <para>
62     /// Gets the header reference.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <returns>
67     /// <para>A ref links header of t link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref *_header;
72 }
73 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLinkAddress = System.UInt64;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 64 external links recursionless size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16    /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
17    public unsafe abstract class UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
    → ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>,
    → ILinksTreeMethods<TLinkAddress>
18    {
19        /// <summary>
20        /// <para>
21        /// The links data parts.
22        /// </para>
23        /// <para></para>

```

```

24     /// </summary>
25     protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
26     /// <summary>
27     /// <para>
28     /// The links index parts.
29     /// </para>
30     /// <para></para>
31     /// </summary>
32     protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
33     /// <summary>
34     /// <para>
35     /// The header.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected new readonly LinksHeader<TLinkAddress>* Header;
40
41     /// <summary>
42     /// <para>
43     /// Initializes a new <see
44     ↪ cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
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 UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
65     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
66     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
67     {
68         LinksDataParts = linksDataParts;
69         LinksIndexParts = linksIndexParts;
70         Header = header;
71     }
72
73     /// <summary>
74     /// <para>
75     /// Gets the zero.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <returns>
80     /// <para>The ulong</para>
81     /// <para></para>
82     /// </returns>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     protected override ulong GetZero() => 0UL;
85
86     /// <summary>
87     /// <para>
88     /// Determines whether this instance equal to zero.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="value">
93     /// <para>The value.</para>
94     /// <para></para>
95     /// </param>
96     /// <returns>
97     /// <para>The bool</para>
98     /// <para></para>
99     /// </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
253     ↪ for ulong
254
255     /// <summary>
256     /// <para>
257     /// Determines whether this instance less than.
258     /// </para>
259     /// </summary>
260     /// <param name="first">
261     /// <para>The first.</para>
262     /// </param>
263     /// <param name="second">
264     /// <para>The second.</para>
265     /// </param>
266     /// <returns>
267     /// <para>The bool</para>
268     /// </returns>
269     [MethodImpl(MethodImplOptions.AggressiveInlining)]
270     protected override bool LessThan(ulong first, ulong second) => first < second;
271
272     /// <summary>
273     /// <para>
274     /// Increments the value.
275     /// </para>
276     /// </summary>
277     /// <param name="value">
278     /// <para>The value.</para>
279     /// </param>
280     /// <returns>
281     /// <para>The ulong</para>
282     /// </returns>
283     [MethodImpl(MethodImplOptions.AggressiveInlining)]
284     protected override ulong Increment(ulong value) => ++value;
285
286     /// <summary>
287     /// <para>
288     /// Decrements the value.
289     /// </para>
290     /// </summary>
291     /// <param name="value">
292     /// <para>The value.</para>
293     /// </param>
294     /// <returns>
295     /// <para>The ulong</para>
296     /// </returns>
297     [MethodImpl(MethodImplOptions.AggressiveInlining)]
298     protected override ulong Decrement(ulong value) => --value;
299
300     /// <summary>
301     /// <para>
302     /// Adds the first.
303     /// </para>
304     /// </summary>
305     /// <param name="first">
306     /// <para>The first.</para>
307     /// </param>
308     /// <param name="second">
309     /// <para>The second.</para>
310     /// </param>
311     /// <returns>
312     /// <para>The ulong</para>
313     /// </returns>
314     [MethodImpl(MethodImplOptions.AggressiveInlining)]
315     protected override ulong Add(ulong first, ulong second) => first + second;
316

```

```

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<TLinkAddress> 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<TLinkAddress>
380    ↪ GetLinkDataPartReference(TLinkAddress link) => 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<TLinkAddress>
398    ↪ GetLinkIndexPartReference(TLinkAddress link) => 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">

```

```

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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress
444         ↪ second)
445     {
446         ref var firstLink = ref LinksDataParts[first];
447         ref var secondLink = ref LinksDataParts[second];
448         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
449             ↪ secondLink.Source, secondLink.Target);
450     }
451 }

```

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 TLinkAddress = 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{TLinkAddress}"/>
16     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
17     public unsafe abstract class UInt64ExternalLinksSizeBalancedTreeMethodsBase :
18         ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
19     {
20         /// <summary>
21         /// <para>
22         /// The links data parts.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
27     }

```

```

27     /// <para>
28     /// The links index parts.
29     /// </para>
30     /// <para></para>
31     /// </summary>
32     protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
33     /// <summary>
34     /// <para>
35     /// The header.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected new readonly LinksHeader<TLinkAddress>* 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<TLinkAddress>
66     ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
67     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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;

```



```

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<TLinkAddress> 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<TLinkAddress>
380     ↪ GetLinkDataPartReference(TLinkAddress link) => 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<TLinkAddress>
398     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
399
400     /// <summary>
401     /// <para>
402     /// Determines whether this instance first is to the left of second.
403     /// </para>
404     /// <para></para>
405     /// </summary>
406     /// <param name="first">
407     /// <para>The first.</para>
408     /// <para></para>
409     /// </param>
410     /// <param name="second">

```

```

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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress
444         ↪ second)
445     {
446         ref var firstLink = ref LinksDataParts[first];
447         ref var secondLink = ref LinksDataParts[second];
448         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
449             ↪ secondLink.Source, secondLink.Target);
450     }
451 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 external links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
16         ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>

```

```

30    /// </param>
31    /// <param name="linksIndexParts">
32    /// <para>A links index parts.</para>
33    /// </para></param>
34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// </para></param>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
    ↪ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
41
42    /// <summary>
43    /// <para>
44    /// Gets the left reference using the specified node.
45    /// </para>
46    /// </para></summary>
47    /// </summary>
48    /// <param name="node">
49    /// <para>The node.</para>
50    /// </para></param>
51    /// </param>
52    /// <returns>
53    /// <para>The ref link</para>
54    /// </para></returns>
55    /// </returns>
56    [MethodImpl(MethodImplOptions.AggressiveInlining)]
57    protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
    ↪ LinksIndexParts[node].LeftAsSource;
58
59    /// <summary>
60    /// <para>
61    /// Gets the right reference using the specified node.
62    /// </para>
63    /// </para></summary>
64    /// </summary>
65    /// <param name="node">
66    /// <para>The node.</para>
67    /// </para></param>
68    /// </param>
69    /// <returns>
70    /// <para>The ref link</para>
71    /// </para></returns>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↪ LinksIndexParts[node].RightAsSource;
75
76    /// <summary>
77    /// <para>
78    /// Gets the left using the specified node.
79    /// </para>
80    /// </para></summary>
81    /// </summary>
82    /// <param name="node">
83    /// <para>The node.</para>
84    /// </para></param>
85    /// </param>
86    /// <returns>
87    /// <para>The link</para>
88    /// </para></returns>
89    /// </returns>
90    [MethodImpl(MethodImplOptions.AggressiveInlining)]
91    protected override TLinkAddress GetLeft(TLinkAddress node) =>
    ↪ LinksIndexParts[node].LeftAsSource;
92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// </para></summary>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// </para></param>

```

```

102     /// </param>
103     /// <returns>
104     /// <para>The link</para>
105     /// <para></para>
106     /// </returns>
107     [MethodImpl(MethodImplOptions.AggressiveInlining)]
108     protected override TLinkAddress GetRight(TLinkAddress node) =>
109         ↪ 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(TLinkAddress node, TLinkAddress left) =>
127         ↪ LinksIndexParts[node].LeftAsSource = left;
128
129     /// <summary>
130     /// <para>
131     /// Sets the right using the specified node.
132     /// </para>
133     /// <para></para>
134     /// </summary>
135     /// <param name="node">
136     /// <para>The node.</para>
137     /// <para></para>
138     /// </param>
139     /// <param name="right">
140     /// <para>The right.</para>
141     /// <para></para>
142     /// </param>
143     [MethodImpl(MethodImplOptions.AggressiveInlining)]
144     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
145         ↪ LinksIndexParts[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 TLinkAddress GetSize(TLinkAddress node) =>
163         ↪ LinksIndexParts[node].SizeAsSource;
164
165     /// <summary>
166     /// <para>
167     /// Sets the size using the specified node.
168     /// </para>
169     /// <para></para>
170     /// </summary>
171     /// <param name="node">
172     /// <para>The node.</para>
173     /// <para></para>
174     /// </param>
175     /// <param name="size">
176     /// <para>The size.</para>
177     /// <para></para>
178     /// </param>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

176     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
177         ↳ LinksIndexParts[node].SizeAsSource = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <returns>
186     /// <para>The link</para>
187     /// <para></para>
188     /// </returns>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override TLinkAddress GetTreeRoot() => Header->RootAsSource;
191
192     /// <summary>
193     /// <para>
194     /// Gets the base part value using the specified node.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="node">
199     /// <para>The node.</para>
200     /// <para></para>
201     /// </param>
202     /// <returns>
203     /// <para>The link</para>
204     /// <para></para>
205     /// </returns>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
208         ↳ LinksDataParts[node].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>
228     /// <param name="secondTarget">
229     /// <para>The second target.</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 FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
238         ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
239         => firstSource < secondSource || firstSource == secondSource && firstTarget <
240         ↳ secondTarget;
241
242     /// <summary>
243     /// <para>
244     /// Determines whether this instance first is to the right of second.
245     /// </para>
246     /// <para></para>
247     /// </summary>
248     /// <param name="firstSource">
249     /// <para>The first source.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="firstTarget">
253     /// <para>The first target.</para>
254     /// <para></para>
255     /// </param>
256     /// <param name="secondSource">
257     /// <para>The second source.</para>
258     /// <para></para>
259     /// </param>
260     /// <param name="secondTarget">
261     /// <para>The second target.</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 FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
270         ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
271         => firstSource > secondSource || firstSource == secondSource && firstTarget >
272         ↳ secondTarget;

```



```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = 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 :
        ↪ UInt64ExternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64ExternalLinksSourcesSizeBalancedTreeMethods"/>
        ↪ 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>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public UInt64ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
    ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    ↪ : base(constants, linksDataParts, linksIndexParts, header) { }

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

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

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

92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// </summary>
98    /// <param name="node">
99    /// <para>The node.</para>
100    /// </para>
101    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// </para>
105    /// </returns>
106

```

```

107 [MethodImpl(MethodImplOptions.AggressiveInlining)]
108 protected override TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress left) =>
    ↳ LinksIndexParts[node].LeftAsSource = left;

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

177
178 /// <summary>

```

```

179    /// <para>
180    /// Gets the tree root.
181    /// </para>
182    /// <para></para>
183    /// </summary>
184    /// <returns>
185    /// <para>The link</para>
186    /// <para></para>
187    /// </returns>
188    [MethodImpl(MethodImplOptions.AggressiveInlining)]
189    protected override TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress node) =>
207        ↳ LinksDataParts[node].Source;
208
209    /// <summary>
210    /// <para>
211    /// Determines whether this instance first is to the left of second.
212    /// </para>
213    /// <para></para>
214    /// </summary>
215    /// <param name="firstSource">
216    /// <para>The first source.</para>
217    /// <para></para>
218    /// </param>
219    /// <param name="firstTarget">
220    /// <para>The first target.</para>
221    /// <para></para>
222    /// </param>
223    /// <param name="secondSource">
224    /// <para>The second source.</para>
225    /// <para></para>
226    /// </param>
227    /// <param name="secondTarget">
228    /// <para>The second target.</para>
229    /// <para></para>
230    /// </param>
231    /// <returns>
232    /// <para>The bool</para>
233    /// <para></para>
234    /// </returns>
235    [MethodImpl(MethodImplOptions.AggressiveInlining)]
236    protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
237        ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
238        => firstSource < secondSource || firstSource == secondSource && firstTarget <
239        ↳ secondTarget;
240
241    /// <summary>
242    /// <para>
243    /// Determines whether this instance first is to the right of second.
244    /// </para>
245    /// <para></para>
246    /// </summary>
247    /// <param name="firstSource">
248    /// <para>The first source.</para>
249    /// <para></para>
250    /// </param>
251    /// <param name="firstTarget">
252    /// <para>The first target.</para>
253    /// <para></para>
254    /// </param>
255    /// <param name="secondSource">
256    /// <para>The second source.</para>
257    /// <para></para>
258    /// </param>
259    /// <returns>
260    /// <para>The bool</para>
261    /// <para></para>
262    /// </returns>
263    [MethodImpl(MethodImplOptions.AggressiveInlining)]
264    protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
265        ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
266        => firstSource > secondSource || firstSource == secondSource && firstTarget >
267        ↳ secondTarget;

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = 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 :
        ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
        ↪ cref="UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
    ↳ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress node) =>
    ↳ LinksIndexParts[node].RightAsTarget;

```

```

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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetSize(TLinkAddress node) =>
162        ↪ LinksIndexParts[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)]
179    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
180        ↪ LinksIndexParts[node].SizeAsTarget = size;
181
182    /// <summary>
183    /// <para>
184    /// Gets the tree root.
185    /// </para>
186    /// <para></para>
187    /// </summary>

```

```

184    /// <returns>
185    /// <para>The link</para>
186    /// <para></para>
187    /// </returns>
188    [MethodImpl(MethodImplOptions.AggressiveInlining)]
189    protected override TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress node) =>
207        ↳ LinksDataParts[node].Target;
208
209    /// <summary>
210    /// <para>
211    /// Determines whether this instance first is to the left of second.
212    /// </para>
213    /// <para></para>
214    /// </summary>
215    /// <param name="firstSource">
216    /// <para>The first source.</para>
217    /// <para></para>
218    /// </param>
219    /// <param name="firstTarget">
220    /// <para>The first target.</para>
221    /// <para></para>
222    /// </param>
223    /// <param name="secondSource">
224    /// <para>The second source.</para>
225    /// <para></para>
226    /// </param>
227    /// <param name="secondTarget">
228    /// <para>The second target.</para>
229    /// <para></para>
230    /// </param>
231    /// <returns>
232    /// <para>The bool</para>
233    /// <para></para>
234    /// </returns>
235    [MethodImpl(MethodImplOptions.AggressiveInlining)]
236    protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
237        ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
238        => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
239        ↳ secondSource;
240
241    /// <summary>
242    /// <para>
243    /// Determines whether this instance first is to the right of second.
244    /// </para>
245    /// <para></para>
246    /// </summary>
247    /// <param name="firstSource">
248    /// <para>The first source.</para>
249    /// <para></para>
250    /// </param>
251    /// <param name="firstTarget">
252    /// <para>The first target.</para>
253    /// <para></para>
254    /// </param>
255    /// <param name="secondSource">
256    /// <para>The second source.</para>
257    /// <para></para>
258    /// </param>
259    /// <param name="secondTarget">
260    /// <para>The second target.</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 FirstIsToRightOfSecond(TLinkAddress firstSource, TLinkAddress
269        ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
270        => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
271        ↳ secondSource;

```



```

259     /// </param>
260     /// <returns>
261     /// <para>The bool</para>
262     /// <para></para>
263     /// </returns>
264     [MethodImpl(MethodImplOptions.AggressiveInlining)]
265     protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
    ↪ => 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(TLinkAddress 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/UInt64ExternalLinksTargetsSizeBalancedTreeMetho

```

1 using System.Runtime.CompilerServices;
2 using TLinkAddress = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 64 external links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64ExternalLinksTargetsSizeBalancedTreeMethods :
    ↪ UInt64ExternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64ExternalLinksTargetsSizeBalancedTreeMethods"/>
    ↪ instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public UInt64ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
    ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetSize(TLinkAddress node) =>
162        ↪ LinksIndexParts[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)]
179    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
180        ↪ LinksIndexParts[node].SizeAsTarget = size;
181
182    /// <summary>
183    /// <para>
184    /// Gets the tree root.
185    /// </para>
186    /// <para></para>
187    /// </summary>
188    /// <returns>
189    /// <para>The link</para>
190    /// <para></para>
191    /// </returns>
192    [MethodImpl(MethodImplOptions.AggressiveInlining)]
193    protected override TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress node) =>
207        ↪ LinksDataParts[node].Target;
208
209    /// <summary>
210    /// <para>
211    /// Determines whether this instance first is to the left of second.
212    /// </para>
213    /// <para></para>
214    /// </summary>
215    /// <param name="firstSource">
216    /// <para>The first source.</para>
217    /// <para></para>
218    /// </param>
219    /// <param name="firstTarget">
220    /// <para>The first target.</para>
221    /// <para></para>
222    /// </param>
223    /// <param name="secondSource">
224    /// <para>The second source.</para>
225    /// <para></para>
226    /// </param>
227    /// <param name="secondTarget">
228    /// <para>The second target.</para>
229    /// <para></para>
230    /// </param>
231    /// <returns>
232    /// <para>The bool</para>
233    /// <para></para>
234    /// </returns>
235    [MethodImpl(MethodImplOptions.AggressiveInlining)]
236    protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
237        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
238        => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
239        ↪ secondSource;
240
241    /// <summary>
242    /// <para>
243    /// Determines whether this instance first is to the right of second.
244    /// </para>
245    /// <para></para>
246    /// </summary>
247    /// <param name="firstSource">
248    /// <para>The first source.</para>
249    /// <para></para>
250    /// </param>
251    /// <param name="firstTarget">
252    /// <para>The first target.</para>
253    /// <para></para>
254    /// </param>
255    /// <param name="secondSource">
256    /// <para>The second source.</para>
257    /// <para></para>
258    /// </param>
259    /// <param name="secondTarget">
260    /// <para>The second target.</para>
261    /// <para></para>
262    /// </param>
263    /// <returns>
264    /// <para>The bool</para>
265    /// <para></para>
266    /// </returns>
267    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

265     protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress 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/UInt64InternalLinksRecursionlessSizeBalancedTreeM

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLinkAddress = 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{TLinkAddress}"/>
16     public unsafe abstract class UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
17     {
18         /// <summary>
19         /// <para>
20         /// The links data parts.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
25         /// <summary>
26         /// <para>
27         /// The links index parts.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
32         /// <summary>
33         /// <para>
34         /// The header.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         protected new readonly LinksHeader<TLinkAddress>* Header;
39
40         /// <summary>
41         /// <para>
42         /// Initializes a new <see
    ↪ cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> 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    /// <param name="header">
59    /// <para>A header.</para>
60    /// <para></para>
61    /// </param>
62    [MethodImpl(MethodImplOptions.AggressiveInlining)]
63    protected UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLi
    ↪ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64    {
65        LinksDataParts = linksDataParts;
66        LinksIndexParts = linksIndexParts;
67        Header = header;
68    }
69
70
71    /// <summary>
72    /// <para>
73    /// Gets the zero.
74    /// </para>
75    /// <para></para>
76    /// </summary>
77    /// <returns>
78    /// <para>The ulong</para>
79    /// <para></para>
80    /// </returns>
81    [MethodImpl(MethodImplOptions.AggressiveInlining)]
82    protected override ulong GetZero() => 0UL;
83
84    /// <summary>
85    /// <para>
86    /// Determines whether this instance equal to zero.
87    /// </para>
88    /// <para></para>
89    /// </summary>
90    /// <param name="value">
91    /// <para>The value.</para>
92    /// <para></para>
93    /// </param>
94    /// <returns>
95    /// <para>The bool</para>
96    /// <para></para>
97    /// </returns>
98    [MethodImpl(MethodImplOptions.AggressiveInlining)]
99    protected override bool EqualToZero(ulong value) => value == 0UL;
100
101    /// <summary>
102    /// <para>
103    /// Determines whether this instance are equal.
104    /// </para>
105    /// <para></para>
106    /// </summary>
107    /// <param name="first">
108    /// <para>The first.</para>
109    /// <para></para>
110    /// </param>
111    /// <param name="second">
112    /// <para>The second.</para>
113    /// <para></para>
114    /// </param>
115    /// <returns>
116    /// <para>The bool</para>
117    /// <para></para>
118    /// </returns>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override bool AreEqual(ulong first, ulong second) => first == second;
121
122    /// <summary>
123    /// <para>
124    /// Determines whether this instance greater than zero.
125    /// </para>
126    /// <para></para>

```

```

127     /// </summary>
128     /// <param name="value">
129     /// <para>The value.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The bool</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected override bool GreaterThanZero(ulong value) => value > 0UL;
138
139     /// <summary>
140     /// <para>
141     /// Determines whether this instance greater than.
142     /// </para>
143     /// <para></para>
144     /// </summary>
145     /// <param name="first">
146     /// <para>The first.</para>
147     /// <para></para>
148     /// </param>
149     /// <param name="second">
150     /// <para>The second.</para>
151     /// <para></para>
152     /// </param>
153     /// <returns>
154     /// <para>The bool</para>
155     /// <para></para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override bool GreaterThan(ulong first, ulong second) => first > second;
159
160     /// <summary>
161     /// <para>
162     /// Determines whether this instance greater or equal than.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     /// <param name="first">
167     /// <para>The first.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="second">
171     /// <para>The second.</para>
172     /// <para></para>
173     /// </param>
174     /// <returns>
175     /// <para>The bool</para>
176     /// <para></para>
177     /// </returns>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
179     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
180
181     /// <summary>
182     /// <para>
183     /// Determines whether this instance greater or equal than zero.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="value">
188     /// <para>The value.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The bool</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
    ↪ 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>
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<TLinkAddress>
366     ↪ GetLinkDataPartReference(TLinkAddress link) => 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<TLinkAddress>
384     ↪ GetLinkIndexPartReference(TLinkAddress link) => 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(TLinkAddress first, TLinkAddress
406     ↪ second) => 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(TLinkAddress first, TLinkAddress
428     ↪ second) => GetKeyPartValue(first) > GetKeyPartValue(second);
429 }

```

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 TLinkAddress = 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{TLinkAddress}"/>
16    public unsafe abstract class UInt64InternalLinksSizeBalancedTreeMethodsBase :
17    ↪ InternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>
18    {
19        /// <summary>
20        /// <para>
21        /// The links data parts.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
26        /// <summary>
27        /// <para>
28        /// The links index parts.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
33        /// <summary>
34        /// <para>
35        /// The header.
36        /// </para>
37        /// <para></para>
38        /// </summary>
39        protected new readonly LinksHeader<TLinkAddress>* Header;
40
41        /// <summary>
42        /// <para>
43        /// Initializes a new <see cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
44        ↪ instance.
45        /// </para>
46        /// <para></para>
47        /// </summary>
48        /// <param name="constants">
49        /// <para>A constants.</para>
50        /// <para></para>
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 UInt64InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
66        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
67        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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>

```

```

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
    ↳ 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<TLinkAddress>
366    ↪ GetLinkDataPartReference(TLinkAddress link) => 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<TLinkAddress>
384    ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
385
386    /// <summary>
387    /// <para>

```

```

386     /// Determines whether this instance first is to the left of second.
387     /// </para>
388     /// <para></para>
389     /// </summary>
390     /// <param name="first">
391     /// <para>The first.</para>
392     /// <para></para>
393     /// </param>
394     /// <param name="second">
395     /// <para>The second.</para>
396     /// <para></para>
397     /// </param>
398     /// <returns>
399     /// <para>The bool</para>
400     /// <para></para>
401     /// </returns>
402     [MethodImpl(MethodImplOptions.AggressiveInlining)]
403     protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress
        ↪ 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(TLinkAddress first, TLinkAddress
        ↪ second) => GetKeyPartValue(first) > GetKeyPartValue(second);
425 }
426 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = 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{TLinkAddress}"/>
15     public unsafe class UInt64InternalLinksSourcesLinkedListMethods :
        ↪ InternalLinksSourcesLinkedListMethods<TLinkAddress>
16     {
17         private readonly RawLinkDataPart<TLinkAddress>* _linksDataParts;
18         private readonly RawLinkIndexPart<TLinkAddress>* _linksIndexParts;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt64InternalLinksSourcesLinkedListMethods"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="constants">
27         /// <para>A constants.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="linksDataParts">
31         /// <para>A links data parts.</para>
32         /// <para></para>

```



```

33     /// </param>
34     /// <param name="linksIndexParts">
35     /// <para>A links index parts.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public UInt64InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress>
        ↳ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts)
40         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
41     {
42         _linksDataParts = linksDataParts;
43         _linksIndexParts = linksIndexParts;
44     }
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<TLinkAddress>
        ↳ GetLinkDataPartReference(TLinkAddress 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<TLinkAddress>
        ↳ GetLinkIndexPartReference(TLinkAddress link) => ref _linksIndexParts[link];
79 }
80 }

```

1.74 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods

```

1 using System.Runtime.CompilerServices;
2 using TLinkAddress = 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
        ↳ cref="UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress node) =>
109        ↪ 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(TLinkAddress node, TLinkAddress left) =>
127        ↪ LinksIndexParts[node].LeftAsSource = left;
128
129    /// <summary>
130    /// <para>
131    /// Sets the right using the specified node.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="node">
136    /// <para>The node.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="right">
140    /// <para>The right.</para>
141    /// <para></para>
142    /// </param>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
145        ↪ LinksIndexParts[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 TLinkAddress GetSize(TLinkAddress node) =>
163        ↪ LinksIndexParts[node].SizeAsSource;
164
165    /// <summary>
166    /// <para>
167    /// Sets the size using the specified node.
168    /// </para>
169    /// <para></para>
170    /// </summary>
171    /// <param name="node">
172    /// <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(TLinkAddress node, TLinkAddress size) =>
177        ↪ LinksIndexParts[node].SizeAsSource = 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 TLinkAddress GetTreeRoot(TLinkAddress node) =>
194        ↪ 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 TLinkAddress GetBasePartValue(TLinkAddress node) =>
212        ↪ LinksDataParts[node].Source;
213
214    /// <summary>
215    /// <para>
216    /// Gets the key part value using the specified node.
217    /// </para>
218    /// <para></para>
219    /// </summary>
220    /// <param name="node">
221    /// <para>The node.</para>
222    /// <para></para>
223    /// </param>
224    /// <returns>
225    /// <para>The link</para>
226    /// <para></para>
227    /// </returns>
228    [MethodImpl(MethodImplOptions.AggressiveInlining)]
229    protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
230        ↪ LinksDataParts[node].Target;
231
232    /// <summary>
233    /// <para>
234    /// Clears the node using the specified node.
235    /// </para>
236    /// <para></para>
237    /// </summary>
238    /// <param name="node">
239    /// <para>The node.</para>
240    /// <para></para>
241    /// </param>
242    [MethodImpl(MethodImplOptions.AggressiveInlining)]
243    protected override void ClearNode(TLinkAddress node)
244    {
245        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 TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
        ↪ SearchCore(GetTreeRoot(source), target);
267     }
268 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = 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<TLinkAddress>
        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetSize(TLinkAddress node) =>
162        ↳ LinksIndexParts[node].SizeAsSource;
163
164    /// <summary>
165    /// <para>
166    /// Sets the size using the specified node.
167    /// </para>
168    /// <para></para>
169    /// </summary>
170    /// <param name="node">
171    /// <para>The node.</para>
172    /// <para></para>
173    /// </param>
174    /// <param name="size">
175    /// <para>The size.</para>
176    /// <para></para>
177    /// </param>
178    [MethodImpl(MethodImplOptions.AggressiveInlining)]
179    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
180        ↳ LinksIndexParts[node].SizeAsSource = size;
181
182    /// <summary>
183    /// <para>
184    /// Gets the tree root using the specified node.
185    /// </para>
186    /// <para></para>
187    /// </summary>
188    /// <param name="node">
189    /// <para>The node.</para>
190    /// <para></para>
191    /// </param>
192    /// <returns>
193    /// <para>The link</para>
194    /// <para></para>
195    /// </returns>
196    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```



```
267 }
268 }
```

1.76 ./csharp/Platform.Data.Doublets.Memory.Split.Specific/UInt64InternalLinksTargetsRecursionlessSizeBalanced

```
1 using System.Runtime.CompilerServices;
2 using TLinkAddress = 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>
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 UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
43        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
44        ↪ : base(constants, linksDataParts, 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>
```

```

69     /// <returns>
70     /// <para>The ref ulong</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref ulong GetRightReference(ulong node) => ref
75     ↪ LinksIndexParts[node].RightAsTarget;
76
77     /// <summary>
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 TLinkAddress GetLeft(TLinkAddress node) =>
93     ↪ 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 TLinkAddress GetRight(TLinkAddress node) =>
111    ↪ LinksIndexParts[node].RightAsTarget;
112
113    /// <summary>
114    /// <para>
115    /// Sets the left using the specified node.
116    /// </para>
117    /// <para></para>
118    /// </summary>
119    /// <param name="node">
120    /// <para>The node.</para>
121    /// <para></para>
122    /// </param>
123    /// <param name="left">
124    /// <para>The left.</para>
125    /// <para></para>
126    /// </param>
127    [MethodImpl(MethodImplOptions.AggressiveInlining)]
128    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
129    ↪ LinksIndexParts[node].LeftAsTarget = left;
130
131    /// <summary>
132    /// <para>
133    /// Sets the right using the specified node.
134    /// </para>
135    /// <para></para>
136    /// </summary>
137    /// <param name="node">
138    /// <para>The node.</para>
139    /// <para></para>
140    /// </param>
141    /// <param name="right">
142    /// <para>The right.</para>
143    /// <para></para>
144    /// </param>
145    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

142     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
143         ↳ LinksIndexParts[node].RightAsTarget = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// </param>
153     /// </returns>
154     /// <para>The link</para>
155     /// </returns>
156     [MethodImpl(MethodImplOptions.AggressiveInlining)]
157     protected override TLinkAddress GetSize(TLinkAddress node) =>
158         ↳ LinksIndexParts[node].SizeAsTarget;
159
160     /// <summary>
161     /// <para>
162     /// Sets the size using the specified node.
163     /// </para>
164     /// </summary>
165     /// <param name="node">
166     /// <para>The node.</para>
167     /// </param>
168     /// <param name="size">
169     /// <para>The size.</para>
170     /// </param>
171     /// </returns>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
174         ↳ LinksIndexParts[node].SizeAsTarget = size;
175
176     /// <summary>
177     /// <para>
178     /// Gets the tree root using the specified node.
179     /// </para>
180     /// </summary>
181     /// <param name="node">
182     /// <para>The node.</para>
183     /// </param>
184     /// </returns>
185     /// <para>The link</para>
186     /// </returns>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     protected override TLinkAddress GetTreeRoot(TLinkAddress node) =>
189         ↳ LinksIndexParts[node].RootAsTarget;
190
191     /// <summary>
192     /// <para>
193     /// Gets the base part value using the specified node.
194     /// </para>
195     /// </summary>
196     /// <param name="node">
197     /// <para>The node.</para>
198     /// </param>
199     /// </returns>
200     /// <para>The link</para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
204         ↳ LinksDataParts[node].Target;
205
206     /// <summary>
207     /// <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 TLinkAddress GetKeyPartValue(TLinkAddress node) =>
228         ↪ LinksDataParts[node].Source;
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(TLinkAddress node)
242     {
243         ref var link = ref LinksIndexParts[node];
244         link.LeftAsTarget = Zero;
245         link.RightAsTarget = Zero;
246         link.SizeAsTarget = 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 TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
268         ↪ SearchCore(GetTreeRoot(target), source);
269 }
270 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksTargetsSizeBalancedTreeMethods :
16         ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>

```

```

19     /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
    → 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 UInt64InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
    → constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    → RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    → : base(constants, linksDataParts, linksIndexParts, header) { }

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 ulong</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref ulong GetLeftReference(ulong node) => ref
    → LinksIndexParts[node].LeftAsTarget;

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

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

```

```

91     protected override TLinkAddress GetLeft(TLinkAddress node) =>
92         ↪ LinksIndexParts[node].LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLinkAddress GetRight(TLinkAddress node) =>
110        ↪ 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>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
128        ↪ LinksIndexParts[node].LeftAsTarget = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">
141    /// <para>The right.</para>
142    /// <para></para>
143    /// </param>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
146        ↪ LinksIndexParts[node].RightAsTarget = right;
147
148    /// <summary>
149    /// <para>
150    /// Gets the size using the specified node.
151    /// </para>
152    /// <para></para>
153    /// </summary>
154    /// <param name="node">
155    /// <para>The node.</para>
156    /// <para></para>
157    /// </param>
158    /// <returns>
159    /// <para>The link</para>
160    /// <para></para>
161    /// </returns>
162    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override TLinkAddress GetSize(TLinkAddress node) =>
        ↪ LinksIndexParts[node].SizeAsTarget;

```

```

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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetTreeRoot(TLinkAddress node) =>
195         ↪ LinksIndexParts[node].RootAsTarget;
196
197     /// <summary>
198     /// <para>
199     /// Gets the base part value using the specified node.
200     /// </para>
201     /// <para></para>
202     /// </summary>
203     /// <param name="node">
204     /// <para>The node.</para>
205     /// <para></para>
206     /// </param>
207     /// <returns>
208     /// <para>The link</para>
209     /// <para></para>
210     /// </returns>
211     [MethodImpl(MethodImplOptions.AggressiveInlining)]
212     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
213         ↪ LinksDataParts[node].Target;
214
215     /// <summary>
216     /// <para>
217     /// Gets the key part value using the specified node.
218     /// </para>
219     /// <para></para>
220     /// </summary>
221     /// <param name="node">
222     /// <para>The node.</para>
223     /// <para></para>
224     /// </param>
225     /// <returns>
226     /// <para>The link</para>
227     /// <para></para>
228     /// </returns>
229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
230     protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
231         ↪ LinksDataParts[node].Source;
232
233     /// <summary>
234     /// <para>
235     /// Clears the node using the specified node.
236     /// </para>
237     /// <para></para>
238     /// </summary>
239     /// <param name="node">
240     /// <para>The node.</para>
241     /// <para></para>
242     /// </param>

```

```

237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLinkAddress 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 TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
        ↪ SearchCore(GetTreeRoot(target), source);
267 }
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 TLinkAddress = 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{TLinkAddress}"/>
19     public unsafe class UInt64SplitMemoryLinks : SplitMemoryLinksBase<TLinkAddress>
20     {
21         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalSourceTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalTargetTreeMethods;
24         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalTargetTreeMethods;
25         private LinksHeader<ulong>* _header;
26         private RawLinkDataPart<ulong>* _linksDataParts;
27         private RawLinkIndexPart<ulong>* _linksIndexParts;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         /// <param name="dataMemory">
36         /// <para>A data memory.</para>
37         /// <para></para>
38         /// </param>
39         /// <param name="indexMemory">
40         /// <para>A index memory.</para>
41         /// <para></para>
42         /// </param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }

```



```

45
46    /// <summary>
47    /// <para>
48    /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
49    /// </para>
50    /// <para></para>
51    /// </summary>
52    /// <param name="dataMemory">
53    /// <para>A data memory.</para>
54    /// <para></para>
55    /// </param>
56    /// <param name="indexMemory">
57    /// <para>A index memory.</para>
58    /// <para></para>
59    /// </param>
60    /// <param name="memoryReservationStep">
61    /// <para>A memory reservation step.</para>
62    /// <para></para>
63    /// </param>
64    [MethodImpl(MethodImplOptions.AggressiveInlining)]
65    public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↳ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
        ↳ IndexTreeType.Default, useLinkedList: true) { }
66
67    /// <summary>
68    /// <para>
69    /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
70    /// </para>
71    /// <para></para>
72    /// </summary>
73    /// <param name="dataMemory">
74    /// <para>A data memory.</para>
75    /// <para></para>
76    /// </param>
77    /// <param name="indexMemory">
78    /// <para>A index memory.</para>
79    /// <para></para>
80    /// </param>
81    /// <param name="memoryReservationStep">
82    /// <para>A memory reservation step.</para>
83    /// <para></para>
84    /// </param>
85    /// <param name="constants">
86    /// <para>A constants.</para>
87    /// <para></para>
88    /// </param>
89    [MethodImpl(MethodImplOptions.AggressiveInlining)]
90    public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> 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<TLinkAddress> constants,
        ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↪ memoryReservationStep, constants, useLinkedList)
124     {
125         if (indexTreeType == IndexTreeType.SizeBalancedTree)
126         {
127             _createInternalSourceTreeMethods = () => new
        ↪ UInt64InternalLinksSourcesSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
128             _createExternalSourceTreeMethods = () => new
        ↪ UInt64ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
129             _createInternalTargetTreeMethods = () => new
        ↪ UInt64InternalLinksTargetsSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
130             _createExternalTargetTreeMethods = () => new
        ↪ UInt64ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
131         }
132         else
133         {
134             _createInternalSourceTreeMethods = () => new
        ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
135             _createExternalSourceTreeMethods = () => new
        ↪ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
136             _createInternalTargetTreeMethods = () => new
        ↪ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
137             _createExternalTargetTreeMethods = () => new
        ↪ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
138         }
139         Init(dataMemory, indexMemory);
140     }
141
142     /// <summary>
143     /// <para>
144     /// Sets the pointers using the specified data memory.
145     /// </para>
146     /// <para></para>
147     /// </summary>
148     /// <param name="dataMemory">
149     /// <para>The data memory.</para>
150     /// <para></para>
151     /// </param>
152     /// <param name="indexMemory">
153     /// <para>The index memory.</para>
154     /// <para></para>
155     /// </param>
156     [MethodImpl(MethodImplOptions.AggressiveInlining)]
157     protected override void SetPointers(IResizableDirectMemory dataMemory,
        ↪ IResizableDirectMemory indexMemory)
158     {
159         _linksDataParts = (RawLinkDataPart<TLinkAddress>*)dataMemory.Pointer;
160         _linksIndexParts = (RawLinkIndexPart<TLinkAddress>*)indexMemory.Pointer;
161         _header = (LinksHeader<TLinkAddress>*)indexMemory.Pointer;
162         if (_useLinkedList)
163         {
164             InternalSourcesListMethods = new
        ↪ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
        ↪ _linksIndexParts);
165         }
166         else
167         {
168             InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
169         }
170         ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();

```

```

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

```

```

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

```

```

325     /// </para>
326     /// <para></para>
327     /// </summary>
328     /// <param name="first">
329     /// <para>The first.</para>
330     /// <para></para>
331     /// </param>
332     /// <param name="second">
333     /// <para>The second.</para>
334     /// <para></para>
335     /// </param>
336     /// <returns>
337     /// <para>The bool</para>
338     /// <para></para>
339     /// </returns>
340     [MethodImpl(MethodImplOptions.AggressiveInlining)]
341     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
342
343     /// <summary>
344     /// <para>
345     /// Gets the zero.
346     /// </para>
347     /// <para></para>
348     /// </summary>
349     /// <returns>
350     /// <para>The ulong</para>
351     /// <para></para>
352     /// </returns>
353     [MethodImpl(MethodImplOptions.AggressiveInlining)]
354     protected override ulong GetZero() => 0UL;
355
356     /// <summary>
357     /// <para>
358     /// Gets the one.
359     /// </para>
360     /// <para></para>
361     /// </summary>
362     /// <returns>
363     /// <para>The ulong</para>
364     /// <para></para>
365     /// </returns>
366     [MethodImpl(MethodImplOptions.AggressiveInlining)]
367     protected override ulong GetOne() => 1UL;
368
369     /// <summary>
370     /// <para>
371     /// Converts the to int 64 using the specified value.
372     /// </para>
373     /// <para></para>
374     /// </summary>
375     /// <param name="value">
376     /// <para>The value.</para>
377     /// <para></para>
378     /// </param>
379     /// <returns>
380     /// <para>The long</para>
381     /// <para></para>
382     /// </returns>
383     [MethodImpl(MethodImplOptions.AggressiveInlining)]
384     protected override long ConvertToInt64(ulong value) => (long)value;
385
386     /// <summary>
387     /// <para>
388     /// Converts the to address using the specified value.
389     /// </para>
390     /// <para></para>
391     /// </summary>
392     /// <param name="value">
393     /// <para>The value.</para>
394     /// <para></para>
395     /// </param>
396     /// <returns>
397     /// <para>The ulong</para>
398     /// <para></para>
399     /// </returns>
400     [MethodImpl(MethodImplOptions.AggressiveInlining)]
401     protected override ulong ConvertToAddress(long value) => (ulong)value;
402

```

```

403     /// <summary>
404     /// <para>
405     /// Adds the first.
406     /// </para>
407     /// <para></para>
408     /// </summary>
409     /// <param name="first">
410     /// <para>The first.</para>
411     /// <para></para>
412     /// </param>
413     /// <param name="second">
414     /// <para>The second.</para>
415     /// <para></para>
416     /// </param>
417     /// <returns>
418     /// <para>The ulong</para>
419     /// <para></para>
420     /// </returns>
421     [MethodImpl(MethodImplOptions.AggressiveInlining)]
422     protected override ulong Add(ulong first, ulong second) => first + second;
423
424     /// <summary>
425     /// <para>
426     /// Subtracts the first.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The ulong</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override ulong Subtract(ulong first, ulong second) => first - second;
444
445     /// <summary>
446     /// <para>
447     /// Increments the link.
448     /// </para>
449     /// <para></para>
450     /// </summary>
451     /// <param name="link">
452     /// <para>The link.</para>
453     /// <para></para>
454     /// </param>
455     /// <returns>
456     /// <para>The ulong</para>
457     /// <para></para>
458     /// </returns>
459     [MethodImpl(MethodImplOptions.AggressiveInlining)]
460     protected override ulong Increment(ulong link) => ++link;
461
462     /// <summary>
463     /// <para>
464     /// Decrements the link.
465     /// </para>
466     /// <para></para>
467     /// </summary>
468     /// <param name="link">
469     /// <para>The link.</para>
470     /// <para></para>
471     /// </param>
472     /// <returns>
473     /// <para>The ulong</para>
474     /// <para></para>
475     /// </returns>
476     [MethodImpl(MethodImplOptions.AggressiveInlining)]
477     protected override ulong Decrement(ulong link) => --link;
478 }
479 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLinkAddress = 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{TLinkAddress}"/>
16     public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<TLinkAddress>
17     {
18         private readonly RawLinkDataPart<ulong>* _links;
19         private readonly LinksHeader<ulong>* _header;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt64UnusedLinksListMethods"/> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt64UnusedLinksListMethods(RawLinkDataPart<ulong>* links, LinksHeader<ulong>*
37         ↪ header)
38             : base((byte*)links, (byte*)header)
39         {
40             _links = links;
41             _header = header;
42         }
43
44         /// <summary>
45         /// <para>
46         /// Gets the link data part reference using the specified link.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="link">
51         /// <para>The link.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>A ref raw link data part of t link</para>
56         /// <para></para>
57         /// </returns>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         protected override ref RawLinkDataPart<TLinkAddress>
60         ↪ GetLinkDataPartReference(TLinkAddress link) => ref _links[link];
61
62         /// <summary>
63         /// <para>
64         /// Gets the header reference.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68         /// <returns>
69         /// <para>A ref links header of t link</para>
70         /// <para></para>
71         /// </returns>
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref *_header;
74     }
75 }

```

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{TLinkAddress}"/>
22     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
23     public unsafe abstract class LinksAvlBalancedTreeMethodsBase<TLinkAddress> :
24         ↳ SizedAndThreadedAVLBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
25     {
26         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
27             ↳ = UncheckedConverter<TLinkAddress, long>.Default;
28         private static readonly UncheckedConverter<TLinkAddress, int> _addressToInt32Converter =
29             ↳ UncheckedConverter<TLinkAddress, int>.Default;
30         private static readonly UncheckedConverter<bool, TLinkAddress> _boolToAddressConverter =
31             ↳ UncheckedConverter<bool, TLinkAddress>.Default;
32         private static readonly UncheckedConverter<TLinkAddress, bool> _addressToBoolConverter =
33             ↳ UncheckedConverter<TLinkAddress, bool>.Default;
34         private static readonly UncheckedConverter<int, TLinkAddress> _int32ToAddressConverter =
35             ↳ UncheckedConverter<int, TLinkAddress>.Default;
36
37         /// <summary>
38         /// <para>
39         /// The break.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         protected readonly TLinkAddress Break;
44
45         /// <summary>
46         /// <para>
47         /// The continue.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         protected readonly TLinkAddress Continue;
52
53         /// <summary>
54         /// <para>
55         /// The links.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         protected readonly byte* Links;
60
61         /// <summary>
62         /// <para>
63         /// The header.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         protected readonly byte* Header;
68
69         /// <summary>
70         /// <para>
71         /// Initializes a new <see cref="LinksAvlBalancedTreeMethodsBase"/> instance.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         /// <param name="constants">
76         /// <para>A constants.</para>
77         /// <para></para>
78         /// </param>
79         /// <param name="links">
80         /// <para>A links.</para>
81         /// <para></para>
82         /// </param>
```



```

73     /// </param>
74     /// <param name="header">
75     /// <para>A header.</para>
76     /// <para></para>
77     /// </param>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected LinksAvlBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, byte*
80     ↪ links, 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 TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress 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(TLinkAddress source, TLinkAddress
145    ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);
146
147    /// <summary>
148    /// <para>
149    /// 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(TLinkAddress source, TLinkAddress
    ↪ target, TLinkAddress rootSource, TLinkAddress 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<TLinkAddress> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLinkAddress>>(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<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
    ↪ AsRef<RawLink<TLinkAddress>>(Links + (RawLink<TLinkAddress>.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<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
221    {

```

```

222     ref var link = ref GetLinkReference(linkIndex);
223     return new Link<TLinkAddress>(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(TLinkAddress first, TLinkAddress second)
246 {
247     ref var firstLink = ref GetLinkReference(first);
248     ref var secondLink = ref GetLinkReference(second);
249     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
250         ↪ secondLink.Source, secondLink.Target);
251 }
252
253 /// <summary>
254 /// <para>
255 /// Determines whether this instance first is to the right of second.
256 /// </para>
257 /// <para></para>
258 /// </summary>
259 /// <param name="first">
260 /// <para>The first.</para>
261 /// <para></para>
262 /// </param>
263 /// <param name="second">
264 /// <para>The second.</para>
265 /// <para></para>
266 /// </param>
267 /// <returns>
268 /// <para>The bool</para>
269 /// <para></para>
270 /// </returns>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
273     ↪ second)
274 {
275     ref var firstLink = ref GetLinkReference(first);
276     ref var secondLink = ref GetLinkReference(second);
277     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
278         ↪ secondLink.Source, secondLink.Target);
279 }
280
281 /// <summary>
282 /// <para>
283 /// Gets the size value using the specified value.
284 /// </para>
285 /// <para></para>
286 /// </summary>
287 /// <param name="value">
288 /// <para>The value.</para>
289 /// <para></para>
290 /// </param>
291 /// <returns>
292 /// <para>The link</para>
293 /// <para></para>
294 /// </returns>
295 [MethodImpl(MethodImplOptions.AggressiveInlining)]
296 protected virtual TLinkAddress GetSizeValue(TLinkAddress value) =>
297     ↪ Bit<TLinkAddress>.PartialRead(value, 5, -5);
298
299 /// <summary>

```

```

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

```

```

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

```

```

445     /// </param>
446     /// <returns>
447     /// <para>The sbyte</para>
448     /// <para></para>
449     /// </returns>
450     [MethodImpl(MethodImplOptions.AggressiveInlining)]
451     protected virtual sbyte GetBalanceValue(TLinkAddress storedValue)
452     {
453         unchecked
454         {
455             var value =
456                 ↪ _addressToInt32Converter.Convert(Bit<TLinkAddress>.PartialRead(storedValue,
457                 ↪ 0, 3));
458             value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
459                 ↪ end of sbyte
460             return (sbyte)value;
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 TLinkAddress storedValue, sbyte value)
479     {
480         unchecked
481         {
482             var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
483                 ↪ value & 3);
484             var modified = Bit<TLinkAddress>.PartialWrite(storedValue, packagedValue, 0, 3);
485             storedValue = modified;
486         }
487     }
488     /// <summary>
489     /// <para>
490     /// The zero.
491     /// </para>
492     /// <para></para>
493     /// </summary>
494     public TLinkAddress this[TLinkAddress index]
495     {
496         [MethodImpl(MethodImplOptions.AggressiveInlining)]
497         get
498         {
499             var root = GetTreeRoot();
500             if (GreaterOrEqualThan(index, GetSize(root)))
501             {
502                 return Zero;
503             }
504             while (!EqualToZero(root))
505             {
506                 var left = GetLeftOrDefault(root);
507                 var leftSize = GetSizeOrZero(left);
508                 if (LessThan(index, leftSize))
509                 {
510                     root = left;
511                     continue;
512                 }
513                 if (AreEqual(index, leftSize))
514                 {
515                     return root;
516                 }
517                 root = GetRightOrDefault(root);
518                 index = Subtract(index, Increment(leftSize));
519             }
520             return Zero; // TODO: Impossible situation exception (only if tree structure
521                 ↪ broken)

```

```

519     }
520 }
521
522 /// <summary>
523 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
524   ↳ (концом).
525 /// </summary>
526 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
527 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
528 /// <returns>Индекс искомой связи.</returns>
529 [MethodImpl(MethodImplOptions.AggressiveInlining)]
530 public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
531 {
532     var root = GetTreeRoot();
533     while (!EqualToZero(root))
534     {
535         ref var rootLink = ref GetLinkReference(root);
536         var rootSource = rootLink.Source;
537         var rootTarget = rootLink.Target;
538         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
539             ↳ node.Key < root.Key
540         {
541             root = GetLeftOrDefault(root);
542         }
543         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
544             ↳ node.Key > root.Key
545         {
546             root = GetRightOrDefault(root);
547         }
548         else // node.Key == root.Key
549         {
550             return root;
551         }
552     }
553     return Zero;
554 }
555
556 // TODO: Return indices range instead of references count
557 /// <summary>
558 /// <para>
559 /// Counts the usages using the specified link.
560 /// </para>
561 /// <para></para>
562 /// </summary>
563 /// <param name="link">
564 /// <para>The link.</para>
565 /// <para></para>
566 /// </param>
567 /// <returns>
568 /// <para>The link</para>
569 /// <para></para>
570 /// </returns>
571 [MethodImpl(MethodImplOptions.AggressiveInlining)]
572 public TLinkAddress CountUsages(TLinkAddress link)
573 {
574     var root = GetTreeRoot();
575     var total = GetSize(root);
576     var totalRightIgnore = Zero;
577     while (!EqualToZero(root))
578     {
579         var @base = GetBasePartValue(root);
580         if (LessOrEqualThan(@base, link))
581         {
582             root = GetRightOrDefault(root);
583         }
584         else
585         {
586             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
587             root = GetLeftOrDefault(root);
588         }
589     }
590     root = GetTreeRoot();
591     var totalLeftIgnore = Zero;
592     while (!EqualToZero(root))
593     {
594         var @base = GetBasePartValue(root);
595         if (GreaterOrEqualThan(@base, link))
596         {

```

```

594         root = GetLeftOrDefault(root);
595     }
596     else
597     {
598         totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
599
600         root = GetRightOrDefault(root);
601     }
602 }
603 return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
604 }
605
606 /// <summary>
607 /// <para>
608 /// Eaches the usage using the specified link.
609 /// </para>
610 /// <para></para>
611 /// </summary>
612 /// <param name="link">
613 /// <para>The link.</para>
614 /// <para></para>
615 /// </param>
616 /// <param name="handler">
617 /// <para>The handler.</para>
618 /// <para></para>
619 /// </param>
620 /// <returns>
621 /// <para>The continue.</para>
622 /// <para></para>
623 /// </returns>
624 [MethodImpl(MethodImplOptions.AggressiveInlining)]
625 public TLinkAddress EachUsage(TLinkAddress link, ReadHandler<TLinkAddress>? handler)
626 {
627     var root = GetTreeRoot();
628     if (EqualToZero(root))
629     {
630         return Continue;
631     }
632     TLinkAddress 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(TLinkAddress 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{TLinkAddress}"/>
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class LinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress> :
23     ↪ RecursionlessSizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
24     {
25         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
26         ↪ = UncheckedConverter<TLinkAddress, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLinkAddress Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLinkAddress 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 }

```

```

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<TLinkAddress>
    ↪ 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 TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress 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(TLinkAddress source, TLinkAddress
    ↪ target, TLinkAddress rootSource, TLinkAddress 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(TLinkAddress source, TLinkAddress
    ↪ target, TLinkAddress rootSource, TLinkAddress 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<TLinkAddress> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLinkAddress>>(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<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
    ↪ AsRef<RawLink<TLinkAddress>>(Links + (RawLink<TLinkAddress>.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<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
216 {
217     ref var link = ref GetLinkReference(linkIndex);
218     return new Link<TLinkAddress>(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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress
268     ↪ second)
269 {
270     ref var firstLink = ref GetLinkReference(first);
271     ref var secondLink = ref GetLinkReference(second);
272     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
273         ↪ secondLink.Source, secondLink.Target);
274 }
275
276 /// <summary>
277 /// <para>
278 /// The zero.
279 /// </para>
280 /// <para></para>

```

```

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

```

```

352 /// <para>The link</para>
353 /// <para></para>
354 /// </returns>
355 [MethodImpl(MethodImplOptions.AggressiveInlining)]
356 public TLinkAddress CountUsages(TLinkAddress 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 TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
412     ↳ EachUsageCore(@base, GetTreeRoot(), handler);
413
414 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
415 ↳ low-level MSIL stack.
416 [MethodImpl(MethodImplOptions.AggressiveInlining)]
417 private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
418     ↳ ReadHandler<TLinkAddress>? handler)
419 {
420     var @continue = Continue;
421     if (EqualToZero(link))
422     {
423         return @continue;
424     }
425     var linkBasePart = GetBasePartValue(link);
426     var @break = Break;
427     if (GreaterThan(linkBasePart, @base))
428     {
429         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(TLinkAddress 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{TLinkAddress}"/>
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class LinksSizeBalancedTreeMethodsBase<TLinkAddress> :
23         ↳ SizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>

```

```

23 {
24     private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
25         ↪ = UncheckedConverter<TLinkAddress, long>.Default;
26
27     /// <summary>
28     /// <para>
29     /// The break.
30     /// </para>
31     /// <para></para>
32     /// </summary>
33     protected readonly TLinkAddress Break;
34     /// <summary>
35     /// <para>
36     /// The continue.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     protected readonly TLinkAddress Continue;
41     /// <summary>
42     /// <para>
43     /// The links.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     protected readonly byte* Links;
48     /// <summary>
49     /// <para>
50     /// The header.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     protected readonly byte* Header;
55
56     /// <summary>
57     /// <para>
58     /// Initializes a new <see cref="LinksSizeBalancedTreeMethodsBase"/> instance.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="constants">
63     /// <para>A constants.</para>
64     /// <para></para>
65     /// </param>
66     /// <param name="links">
67     /// <para>A links.</para>
68     /// <para></para>
69     /// </param>
70     /// <param name="header">
71     /// <para>A header.</para>
72     /// <para></para>
73     /// </param>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected LinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, byte*
76         ↪ links, byte* header)
77     {
78         Links = links;
79         Header = header;
80         Break = constants.Break;
81         Continue = constants.Continue;
82     }
83
84     /// <summary>
85     /// <para>
86     /// Gets the tree root.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <returns>
91     /// <para>The link</para>
92     /// <para></para>
93     /// </returns>
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     protected abstract TLinkAddress GetTreeRoot();
96
97     /// <summary>
98     /// <para>
99     /// Gets the base part value using the specified link.
100    /// </para>
101    /// <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 TLinkAddress GetBasePartValue(TLinkAddress link);
111
112     /// <summary>
113     /// <para>
114     /// <para>Determines whether this instance first is to the right of second.
115     /// </para>
116     /// <para></para>
117     /// </summary>
118     /// <param name="source">
119     /// <para>The source.</para>
120     /// <para></para>
121     /// </param>
122     /// <param name="target">
123     /// <para>The target.</para>
124     /// <para></para>
125     /// </param>
126     /// <param name="rootSource">
127     /// <para>The root source.</para>
128     /// <para></para>
129     /// </param>
130     /// <param name="rootTarget">
131     /// <para>The root target.</para>
132     /// <para></para>
133     /// </param>
134     /// <returns>
135     /// <para>The bool</para>
136     /// <para></para>
137     /// </returns>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     protected abstract bool FirstIsToTheRightOfSecond(TLinkAddress source, TLinkAddress
140     ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);
141
142     /// <summary>
143     /// <para>
144     /// <para>Determines whether this instance first is to the left of second.
145     /// </para>
146     /// <para></para>
147     /// </summary>
148     /// <param name="source">
149     /// <para>The source.</para>
150     /// <para></para>
151     /// </param>
152     /// <param name="target">
153     /// <para>The target.</para>
154     /// <para></para>
155     /// </param>
156     /// <param name="rootSource">
157     /// <para>The root source.</para>
158     /// <para></para>
159     /// </param>
160     /// <param name="rootTarget">
161     /// <para>The root target.</para>
162     /// <para></para>
163     /// </param>
164     /// <returns>
165     /// <para>The bool</para>
166     /// <para></para>
167     /// </returns>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     protected abstract bool FirstIsToTheLeftOfSecond(TLinkAddress source, TLinkAddress
170     ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);
171
172     /// <summary>
173     /// <para>
174     /// <para>Gets the header reference.
175     /// </para>
176     /// <para></para>
177     /// </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<TLinkAddress> GetHeaderReference() => ref
        ↳ AsRef<LinksHeader<TLinkAddress>>(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<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
        ↳ AsRef<RawLink<TLinkAddress>>(Links + (RawLink<TLinkAddress>.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<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
216    {
217        ref var link = ref GetLinkReference(linkIndex);
218        return new Link<TLinkAddress>(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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress
    ↪ second)
267     {
268         ref var firstLink = ref GetLinkReference(first);
269         ref var secondLink = ref GetLinkReference(second);
270         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
271     }
272
273     /// <summary>
274     /// <para>
275     /// The zero.
276     /// </para>
277     /// <para></para>
278     /// </summary>
279     public TLinkAddress this[TLinkAddress 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
    ↪ 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 TLinkAddress Search(TLinkAddress source, TLinkAddress 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)) //
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 TLinkAddress CountUsages(TLinkAddress 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 TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? 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 TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
        ↳ ReadHandler<TLinkAddress>? 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(TLinkAddress 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{TLinkAddress}"/>
14     public unsafe class LinksSourcesAvlBalancedTreeMethods<TLinkAddress> :
15         ↳ LinksAvlBalancedTreeMethodsBase<TLinkAddress>
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<TLinkAddress> constants, byte*
37             ↳ links, 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 TLinkAddress GetLeftReference(TLinkAddress node) => ref
55             ↳ GetLinkReference(node).LeftAsSource;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref link</para>
69         /// <para></para>
70         /// </returns>

```

```

67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↪ GetLinkReference(node).RightAsSource;

70
71     /// <summary>
72     /// <para>
73     /// Gets the left using the specified node.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="node">
78     /// <para>The node.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLinkAddress GetLeft(TLinkAddress node) =>
    ↪ GetLinkReference(node).LeftAsSource;

87
88     /// <summary>
89     /// <para>
90     /// Gets the right using the specified node.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="node">
95     /// <para>The node.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress right) =>
    ↪ GetLinkReference(node).RightAsSource = right;

```

```

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

```



```

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(TLinkAddress node) =>
223         ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsSource);
224
225     /// <summary>
226     /// <para>
227     /// Sets the right is child using the specified node.
228     /// </para>
229     /// <para></para>
230     /// </summary>
231     /// <param name="node">
232     /// <para>The node.</para>
233     /// <para></para>
234     /// </param>
235     /// <param name="value">
236     /// <para>The value.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void SetRightIsChild(TLinkAddress node, bool value) =>
241         ↪ SetRightIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
242
243     /// <summary>
244     /// <para>
245     /// Gets the balance using the specified node.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     /// <param name="node">
250     /// <para>The node.</para>
251     /// <para></para>
252     /// </param>
253     /// <returns>
254     /// <para>The sbyte</para>
255     /// <para></para>
256     /// </returns>
257     [MethodImpl(MethodImplOptions.AggressiveInlining)]
258     protected override sbyte GetBalance(TLinkAddress node) =>
259         ↪ GetBalanceValue(GetLinkReference(node).SizeAsSource);
260
261     /// <summary>
262     /// <para>
263     /// Sets the balance using the specified node.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="node">
268     /// <para>The node.</para>
269     /// <para></para>
270     /// </param>
271     /// <param name="value">
272     /// <para>The value.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void SetBalance(TLinkAddress node, sbyte value) =>
277         ↪ SetBalanceValue(ref GetLinkReference(node).SizeAsSource, value);
278
279     /// <summary>
280     /// <para>
281     /// Gets the tree root.
282     /// </para>
283     /// <para></para>
284     /// </summary>
285     /// <returns>
286     /// <para>The link</para>
287     /// <para></para>
288     /// </returns>
289     [MethodImpl(MethodImplOptions.AggressiveInlining)]
290     protected override TLinkAddress 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 TLinkAddress GetBasePartValue(TLinkAddress link) =>
304         ↪ GetLinkReference(link).Source;
305
306     /// <summary>
307     /// <para>
308     /// Determines whether this instance first is to the left of second.
309     /// </para>
310     /// <para></para>
311     /// </summary>
312     /// <param name="firstSource">
313     /// <para>The first source.</para>
314     /// <para></para>
315     /// </param>
316     /// <param name="firstTarget">
317     /// <para>The first target.</para>
318     /// <para></para>
319     /// </param>
320     /// <param name="secondSource">
321     /// <para>The second source.</para>
322     /// <para></para>
323     /// </param>
324     /// <param name="secondTarget">
325     /// <para>The second target.</para>
326     /// <para></para>
327     /// </param>
328     /// <returns>
329     /// <para>The bool</para>
330     /// <para></para>
331     /// </returns>
332     [MethodImpl(MethodImplOptions.AggressiveInlining)]
333     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
334         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
335         ↪ LessThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
336         ↪ LessThan(firstTarget, secondTarget));
337
338     /// <summary>
339     /// <para>
340     /// Determines whether this instance first is to the right of second.
341     /// </para>
342     /// <para></para>
343     /// </summary>
344     /// <param name="firstSource">
345     /// <para>The first source.</para>
346     /// <para></para>
347     /// </param>
348     /// <param name="firstTarget">
349     /// <para>The first target.</para>
350     /// <para></para>
351     /// </param>
352     /// <param name="secondSource">
353     /// <para>The second source.</para>
354     /// <para></para>
355     /// </param>
356     /// <param name="secondTarget">
357     /// <para>The second target.</para>
358     /// <para></para>
359     /// </param>
360     /// <returns>
361     /// <para>The bool</para>
362     /// <para></para>
363     /// </returns>

```

```

360 [MethodImpl(MethodImplOptions.AggressiveInlining)]
361 protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
    ↪ GreaterThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
    ↪ GreaterThan(firstTarget, secondTarget));
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(TLinkAddress node)
375 {
376     ref var link = ref GetLinkReference(node);
377     link.LeftAsSource = Zero;
378     link.RightAsSource = Zero;
379     link.SizeAsSource = Zero;
380 }
381 }
382 }

```

1.84 ./csharp/Platform.Data.Doublets/Memory/United/Generic/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{TLinkAddress}"/>
14    public unsafe class LinksSourcesRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
    ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="LinksSourcesRecursionlessSizeBalancedTreeMethods"/>
    ↪ 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 LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
    ↪ 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 TLinkAddress GetLeftReference(TLinkAddress node) => ref
    ↪ GetLinkReference(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 link</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↪ GetLinkReference(node).RightAsSource;

70
71     /// <summary>
72     /// <para>
73     /// Gets the left using the specified node.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="node">
78     /// <para>The node.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLinkAddress GetLeft(TLinkAddress node) =>
    ↪ GetLinkReference(node).LeftAsSource;

87
88     /// <summary>
89     /// <para>
90     /// Gets the right using the specified node.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="node">
95     /// <para>The node.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress left) =>
    ↪ GetLinkReference(node).LeftAsSource = 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(TLinkAddress node, TLinkAddress right) =>
137     ↪ 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 TLinkAddress GetSize(TLinkAddress node) =>
155     ↪ GetLinkReference(node).SizeAsSource;
156
157     /// <summary>
158     /// <para>
159     /// Sets the size using the specified node.
160     /// </para>
161     /// <para></para>
162     /// </summary>
163     /// <param name="node">
164     /// <para>The node.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="size">
168     /// <para>The size.</para>
169     /// <para></para>
170     /// </param>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
173     ↪ GetLinkReference(node).SizeAsSource = size;
174
175     /// <summary>
176     /// <para>
177     /// Gets the tree root.
178     /// </para>
179     /// <para></para>
180     /// </summary>
181     /// <returns>
182     /// <para>The link</para>
183     /// <para></para>
184     /// </returns>
185     [MethodImpl(MethodImplOptions.AggressiveInlining)]
186     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsSource;
187
188     /// <summary>
189     /// <para>
190     /// Gets the base part value using the specified link.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     /// <param name="link">
195     /// <para>The link.</para>
196     /// <para></para>
197     /// </param>

```

```

196     /// <returns>
197     /// <para>The link</para>
198     /// <para></para>
199     /// </returns>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
202         ↪ GetLinkReference(link).Source;
203
204     /// <summary>
205     /// <para>
206     /// Determines whether this instance first is to the left of second.
207     /// </para>
208     /// <para></para>
209     /// </summary>
210     /// <param name="firstSource">
211     /// <para>The first source.</para>
212     /// <para></para>
213     /// </param>
214     /// <param name="firstTarget">
215     /// <para>The first target.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="secondSource">
219     /// <para>The second source.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondTarget">
223     /// <para>The second target.</para>
224     /// <para></para>
225     /// </param>
226     /// <returns>
227     /// <para>The bool</para>
228     /// <para></para>
229     /// </returns>
230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
231     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
232         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
233         ↪ LessThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
234         ↪ 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(TLinkAddress firstSource, TLinkAddress
264         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
265         ↪ GreaterThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
266         ↪ GreaterThan(firstTarget, secondTarget));
267
268     /// <summary>
269     /// <para>
270     /// Clears the node using the specified node.
271     /// </para>
272     /// <para></para>
273     /// </summary>

```

```

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(TLinkAddress 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{TLinkAddress}"/>
14     public unsafe class LinksSourcesSizeBalancedTreeMethods<TLinkAddress> :
15         ↳ LinksSizeBalancedTreeMethodsBase<TLinkAddress>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksSourcesSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public LinksSourcesSizeBalancedTreeMethods(LinksConstants<TLinkAddress> constants, byte*
37             ↳ links, 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 TLinkAddress GetLeftReference(TLinkAddress node) => ref
55             ↳ GetLinkReference(node).LeftAsSource;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref link</para>
69         /// <para></para>
70         /// </returns>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
73             ↳ GetLinkReference(node).RightAsSource;
74     }
75 }

```

```

59     /// </summary>
60     /// <param name="node">
61     /// <para>The node.</para>
62     /// <para></para>
63     /// </param>
64     /// <returns>
65     /// <para>The ref link</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↪ GetLinkReference(node).RightAsSource;

70
71     /// <summary>
72     /// <para>
73     /// Gets the left using the specified node.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="node">
78     /// <para>The node.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLinkAddress GetLeft(TLinkAddress node) =>
    ↪ GetLinkReference(node).LeftAsSource;

87
88     /// <summary>
89     /// <para>
90     /// Gets the right using the specified node.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="node">
95     /// <para>The node.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress right) =>
138         ↪ GetLinkReference(node).RightAsSource = right;
139
140     /// <summary>
141     /// <para>
142     /// Gets the size using the specified node.
143     /// </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 TLinkAddress GetSize(TLinkAddress node) =>
155         ↪ GetLinkReference(node).SizeAsSource;
156
157     /// <summary>
158     /// <para>
159     /// Sets the size using the specified node.
160     /// </para>
161     /// <para></para>
162     /// </summary>
163     /// <param name="node">
164     /// <para>The node.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="size">
168     /// <para>The size.</para>
169     /// <para></para>
170     /// </param>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
173         ↪ GetLinkReference(node).SizeAsSource = size;
174
175     /// <summary>
176     /// <para>
177     /// Gets the tree root.
178     /// </para>
179     /// <para></para>
180     /// </summary>
181     /// <returns>
182     /// <para>The link</para>
183     /// <para></para>
184     /// </returns>
185     [MethodImpl(MethodImplOptions.AggressiveInlining)]
186     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsSource;
187
188     /// <summary>
189     /// <para>
190     /// Gets the base part value using the specified link.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     /// <param name="link">
195     /// <para>The link.</para>
196     /// <para></para>
197     /// </param>
198     /// <returns>
199     /// <para>The link</para>
200     /// <para></para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
204         ↪ GetLinkReference(link).Source;
205
206     /// <summary>
207     /// <para>
208     /// Determines whether this instance first is to the left of second.
209     /// </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(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress 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{TLinkAddress}"/>
14    public unsafe class LinksTargetsAvlBalancedTreeMethods<TLinkAddress> :
15        ↳ LinksAvlBalancedTreeMethodsBase<TLinkAddress>
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        /// </param>
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 LinksTargetsAvlBalancedTreeMethods(LinksConstants<TLinkAddress> constants, byte*
34            ↳ links, byte* 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        /// </param>
45        /// <returns>
46        /// <para>The ref link</para>
47        /// </returns>
48        [MethodImpl(MethodImplOptions.AggressiveInlining)]
49        protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
50            ↳ GetLinkReference(node).LeftAsTarget;
51
52        /// <summary>
53        /// <para>
54        /// Gets the right reference using the specified node.
55        /// </para>
56        /// <para></para>
57        /// </summary>
58        /// <param name="node">
59        /// <para>The node.</para>
60        /// </param>
61        /// <returns>
62        /// <para>The ref link</para>
63        /// </returns>
64        [MethodImpl(MethodImplOptions.AggressiveInlining)]
65        protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
66            ↳ GetLinkReference(node).RightAsTarget;
67
68    }
69 }
```

```

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 TLinkAddress GetLeft(TLinkAddress node) =>
87        ↪ GetLinkReference(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 link</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected override TLinkAddress GetRight(TLinkAddress node) =>
105        ↪ 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(TLinkAddress node, TLinkAddress left) =>
123        ↪ GetLinkReference(node).LeftAsTarget = left;
124
125    /// <summary>
126    /// <para>
127    /// Sets the right using the specified node.
128    /// </para>
129    /// <para></para>
130    /// </summary>
131    /// <param name="node">
132    /// <para>The node.</para>
133    /// <para></para>
134    /// </param>
135    /// <param name="right">
136    /// <para>The right.</para>
137    /// <para></para>
138    /// </param>
139    [MethodImpl(MethodImplOptions.AggressiveInlining)]
140    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
141        ↪ GetLinkReference(node).RightAsTarget = right;
142
143    /// <summary>
144    /// <para>
145    /// Gets the size using the specified node.
146    /// </para>
147    /// <para></para>
148    /// </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 TLinkAddress GetSize(TLinkAddress node) =>
155         ↪ GetSizeValue(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(TLinkAddress node, TLinkAddress size) =>
173         ↪ SetSizeValue(ref GetLinkReference(node).SizeAsTarget, size);
174
175     /// <summary>
176     /// <para>
177     /// Determines whether this instance get left is child.
178     /// </para>
179     /// <para></para>
180     /// </summary>
181     /// <param name="node">
182     /// <para>The node.</para>
183     /// <para></para>
184     /// </param>
185     /// <returns>
186     /// <para>The bool</para>
187     /// <para></para>
188     /// </returns>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override bool GetLeftIsChild(TLinkAddress node) =>
191         ↪ GetLeftIsChildValue(GetLinkReference(node).SizeAsTarget);
192
193     /// <summary>
194     /// <para>
195     /// Sets the left is child using the specified node.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="node">
200     /// <para>The node.</para>
201     /// <para></para>
202     /// </param>
203     /// <param name="value">
204     /// <para>The value.</para>
205     /// <para></para>
206     /// </param>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override void SetLeftIsChild(TLinkAddress node, bool value) =>
209         ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
210
211     /// <summary>
212     /// <para>
213     /// Determines whether this instance get right is child.
214     /// </para>
215     /// <para></para>
216     /// </summary>
217     /// <param name="node">
218     /// <para>The node.</para>
219     /// <para></para>
220     /// </param>
221     /// <returns>
222     /// <para>The bool</para>
223     /// <para></para>
224     /// </returns>

```

```

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

```

```

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

```

```

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(TLinkAddress 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

```

using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the links targets recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
    public unsafe class LinksTargetsRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
        LinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksTargetsRecursionlessSizeBalancedTreeMethods"/>
        /// instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
            constants, byte* links, byte* header) : base(constants, links, 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 TLinkAddress GetLeftReference(TLinkAddress node) => ref
            GetLinkReference(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 link</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ref TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress 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 TLinkAddress GetSize(TLinkAddress node) =>
156         ↪ GetLinkReference(node).SizeAsTarget;
157
158     /// <summary>
159     /// <para>
160     /// Sets the size using the specified node.
161     /// </para>
162     /// <para></para>
163     /// </summary>
164     /// <param name="node">
165     /// <para>The node.</para>
166     /// <para></para>
167     /// </param>
168     /// <param name="size">
169     /// <para>The size.</para>
170     /// <para></para>
171     /// </param>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
174         ↪ GetLinkReference(node).SizeAsTarget = size;
175
176     /// <summary>
177     /// <para>
178     /// Gets the tree root.
179     /// </para>
180     /// <para></para>
181     /// </summary>
182     /// <returns>
183     /// <para>The link</para>
184     /// <para></para>
185     /// </returns>
186     [MethodImpl(MethodImplOptions.AggressiveInlining)]
187     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsTarget;
188
189     /// <summary>
190     /// <para>
191     /// Gets the base part value using the specified link.
192     /// </para>
193     /// <para></para>
194     /// </summary>
195     /// <param name="link">
196     /// <para>The link.</para>
197     /// <para></para>
198     /// </param>
199     /// <returns>
200     /// <para>The link</para>
201     /// <para></para>
202     /// </returns>
203     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

201     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
202         ↪ GetLinkReference(link).Target;
203
204     /// <summary>
205     /// <para>
206     /// Determines whether this instance first is to the left of second.
207     /// </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(TLinkAddress firstSource, TLinkAddress
231         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
232         ↪ LessThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
233         ↪ LessThan(firstSource, secondSource));
234
235     /// <summary>
236     /// <para>
237     /// Determines whether this instance first is to the right of second.
238     /// </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(TLinkAddress firstSource, TLinkAddress
262         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
263         ↪ GreaterThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
264         ↪ GreaterThan(firstSource, secondSource));
265
266     /// <summary>
267     /// <para>
268     /// Clears the node using the specified node.
269     /// </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(TLinkAddress 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{TLinkAddress}"/>
14     public unsafe class LinksTargetsSizeBalancedTreeMethods<TLinkAddress> :
15         ↳ LinksSizeBalancedTreeMethodsBase<TLinkAddress>
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<TLinkAddress> constants, byte*
37             ↳ links, 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 TLinkAddress GetLeftReference(TLinkAddress 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 TLinkAddress GetRightReference(TLinkAddress 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 TLinkAddress GetLeft(TLinkAddress 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 TLinkAddress GetRight(TLinkAddress 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(TLinkAddress node, TLinkAddress 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(TLinkAddress node, TLinkAddress right) =>
138         ↪ GetLinkReference(node).RightAsTarget = right;
139
140     /// <summary>
141     /// <para>
142     /// Gets the size using the specified node.
143     /// </para>
144     /// </summary>
145     /// <param name="node">
146     /// <para>The node.</para>
147     /// </param>
148     /// </returns>
149     /// <para>The link</para>
150     /// </returns>
151     [MethodImpl(MethodImplOptions.AggressiveInlining)]
152     protected override TLinkAddress GetSize(TLinkAddress node) =>
153         ↪ GetLinkReference(node).SizeAsTarget;
154
155     /// <summary>
156     /// <para>
157     /// Sets the size using the specified node.
158     /// </para>
159     /// </summary>
160     /// <param name="node">
161     /// <para>The node.</para>
162     /// </param>
163     /// <param name="size">
164     /// <para>The size.</para>
165     /// </param>
166     /// </returns>
167     [MethodImpl(MethodImplOptions.AggressiveInlining)]
168     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
169         ↪ GetLinkReference(node).SizeAsTarget = size;
170
171     /// <summary>
172     /// <para>
173     /// Gets the tree root.
174     /// </para>
175     /// </summary>
176     /// <returns>
177     /// <para>The link</para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsTarget;
181
182     /// <summary>
183     /// <para>
184     /// Gets the base part value using the specified link.
185     /// </para>
186     /// </summary>
187     /// <param name="link">
188     /// <para>The link.</para>
189     /// </param>
190     /// </returns>
191     /// <para>The link</para>
192     /// </returns>
193     [MethodImpl(MethodImplOptions.AggressiveInlining)]
194     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
195         ↪ GetLinkReference(link).Target;
196
197     /// <summary>
198     /// <para>
199     /// Determines whether this instance first is to the left of second.
200     /// </para>
201     /// </summary>
202     /// <param name="firstSource">
203     /// <para>The first source.</para>
204     /// </param>

```

```

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(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress 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(TLinkAddress 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{TLinkAddress}"/>
18     public unsafe class UnitedMemoryLinks<TLinkAddress> : UnitedMemoryLinksBase<TLinkAddress>
19     {
20         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createSourceTreeMethods;
21         private readonly Func<ILinksTreeMethods<TLinkAddress>> _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         {
48             FileMappedResizableDirectMemory(address, memoryReservationStep),
49             memoryReservationStep
50         }) { }
51
52         /// <summary>
53         /// <para>
54         /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <param name="memory">
59         /// <para>A memory.</para>
60         /// <para></para>
61         /// </param>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
64         {
65             DefaultLinksSizeStep
66         }) { }
67
68         /// <summary>
69         /// <para>
70         /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <param name="memory">
75         /// <para>A memory.</para>
76         /// <para></para>
77         /// </param>
78         /// <param name="memoryReservationStep">
79         /// <para>A memory reservation step.</para>
80         /// <para></para>
81         /// </param>
82     }
83 }

```



```

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

```

```

140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 protected override void ResetPointers()
142 {
143     base.ResetPointers();
144     _links = null;
145     _header = null;
146 }
147
148 /// <summary>
149 /// <para>
150 /// 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<TLinkAddress> GetHeaderReference() => ref
    ↳ AsRef<LinksHeader<TLinkAddress>>(_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<TLinkAddress> GetLinkReference(TLinkAddress linkIndex) =>
    ↳ ref AsRef<RawLink<TLinkAddress>>(_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{TLinkAddress}"/>
24     public abstract class UnitedMemoryLinksBase<TLinkAddress> : DisposableBase,
        ↳ ILinks<TLinkAddress> where TLinkAddress : struct
25     {
26         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
            ↳ EqualityComparer<TLinkAddress>.Default;
27         private static readonly Comparer<TLinkAddress> _comparer =
            ↳ Comparer<TLinkAddress>.Default;
28         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
            ↳ = UncheckedConverter<TLinkAddress, long>.Default;
29         private static readonly UncheckedConverter<long, TLinkAddress> _int64ToAddressConverter
            ↳ = UncheckedConverter<long, TLinkAddress>.Default;
30         private static readonly TLinkAddress _zero = default;

```

```

31 private static readonly TLinkAddress _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<TLinkAddress>.SizeInBytes;
39
40 /// <summary>
41 /// <para>
42 /// The size in bytes.
43 /// </para>
44 /// <para></para>
45 /// </summary>
46 public static readonly long LinkHeaderSizeInBytes =
47     ↳ LinksHeader<TLinkAddress>.SizeInBytes;
48
49 /// <summary>
50 /// <para>
51 /// The link size in bytes.
52 /// </para>
53 /// <para></para>
54 /// </summary>
55 public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
56
57 /// <summary>
58 /// <para>
59 /// The memory.
60 /// </para>
61 /// <para></para>
62 /// </summary>
63 protected readonly IResizableDirectMemory _memory;
64
65 /// <summary>
66 /// <para>
67 /// The memory reservation step.
68 /// </para>
69 /// <para></para>
70 /// </summary>
71 protected readonly long _memoryReservationStep;
72
73 /// <summary>
74 /// <para>
75 /// The targets tree methods.
76 /// </para>
77 /// <para></para>
78 /// </summary>
79 protected ILinksTreeMethods<TLinkAddress> TargetsTreeMethods;
80
81 /// <summary>
82 /// <para>
83 /// The sources tree methods.
84 /// </para>
85 /// <para></para>
86 /// </summary>
87 protected ILinksTreeMethods<TLinkAddress> SourcesTreeMethods;
88
89 // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
90 // ↳ нужно использовать не список а дерево, так как так можно быстрее проверить на
91 // ↳ наличие связи внутри
92 /// <summary>
93 /// <para>
94 /// The unused links list methods.
95 /// </para>
96 /// <para></para>
97 /// </summary>
98 protected ILinksListMethods<TLinkAddress> UnusedLinksListMethods;
99
100 /// <summary>
101 /// Возвращает общее число связей находящихся в хранилище.
102 /// </summary>
103 protected virtual TLinkAddress Total
104 {
105     [MethodImpl(MethodImplOptions.AggressiveInlining)]
106     get
107     {
108         ref var header = ref GetHeaderReference();
109         return Subtract(header.AllocatedLinks, header.FreeLinks);
110     }
111 }

```

```

107     /// <summary>
108     /// <para>
109     /// Gets the constants value.
110     /// </para>
111     /// <para></para>
112     /// </summary>
113     public virtual LinksConstants<TLinkAddress> Constants
114     {
115         [MethodImpl(MethodImplOptions.AggressiveInlining)]
116         get;
117     }
118
119     /// <summary>
120     /// <para>
121     /// 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
139     ↪ memoryReservationStep, LinksConstants<TLinkAddress> constants)
140     {
141         _memory = memory;
142         _memoryReservationStep = memoryReservationStep;
143         Constants = constants;
144     }
145
146     /// <summary>
147     /// <para>
148     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="memory">
153     /// <para>A memory.</para>
154     /// <para></para>
155     /// </param>
156     /// <param name="memoryReservationStep">
157     /// <para>A memory reservation step.</para>
158     /// <para></para>
159     /// </param>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
162     ↪ memoryReservationStep) : this(memory, memoryReservationStep,
163     ↪ Default<LinksConstants<TLinkAddress>>.Instance) { }
164
165     /// <summary>
166     /// <para>
167     /// Inits the memory.
168     /// </para>
169     /// <para></para>
170     /// </summary>
171     /// <param name="memory">
172     /// <para>The memory.</para>
173     /// <para></para>
174     /// </param>
175     /// <param name="memoryReservationStep">
176     /// <para>The memory reservation step.</para>
177     /// <para></para>
178     /// </param>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
181     {
182         if (memory.ReservedCapacity < memoryReservationStep)
183         {
184             memory.ReservedCapacity = memoryReservationStep;
185         }
186     }

```

```

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 TLinkAddress Count(ICollection<TLinkAddress>? 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 }

```

```

256     }
257     if (restriction.Count == 3)
258     {
259         var source = restriction[constants.SourcePart];
260         var target = restriction[constants.TargetPart];
261         if (AreEqual(index, any))
262         {
263             if (AreEqual(source, any) && AreEqual(target, any))
264             {
265                 return Total;
266             }
267             else if (AreEqual(source, any))
268             {
269                 return TargetsTreeMethods.CountUsages(target);
270             }
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(TLinkAddress);
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     throw new NotSupportedException("Другие размеры и способы ограничений не
320     ⇨ поддерживаются.");
321 }
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">

```

```

331 /// <para>The substitution.</para>
332 /// <para></para>
333 /// </param>
334 /// <exception cref="NotSupportedException">
335 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
336 /// <para></para>
337 /// </exception>
338 /// <returns>
339 /// <para>The link</para>
340 /// <para></para>
341 /// </returns>
342 [MethodImpl(MethodImplOptions.AggressiveInlining)]
343 public virtual TLinkAddress Each(IList<TLinkAddress>? restriction,
    ↳ ReadHandler<TLinkAddress>? handler)
344 {
345     var constants = Constants;
346     var @break = constants.Break;
347     if (restriction.Count == 0)
348     {
349         for (var link = GetOne(); LessOrEqualThan(link,
    ↳ GetHeaderReference().AllocatedLinks); link = Increment(link))
350         {
351             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
352             {
353                 return @break;
354             }
355         }
356         return @break;
357     }
358     var @continue = constants.Continue;
359     var any = constants.Any;
360     var index = restriction[constants.IndexPart];
361     if (restriction.Count == 1)
362     {
363         if (AreEqual(index, any))
364         {
365             return Each(Array.Empty<TLinkAddress>(), handler);
366         }
367         if (!Exists(index))
368         {
369             return @continue;
370         }
371         return handler(GetLinkStruct(index));
372     }
373     if (restriction.Count == 2)
374     {
375         var value = restriction[1];
376         if (AreEqual(index, any))
377         {
378             if (AreEqual(value, any))
379             {
380                 return Each(Array.Empty<TLinkAddress>(), handler);
381             }
382             if (AreEqual(Each(new Link<TLinkAddress>(index, value, any), handler),
    ↳ @break))
383             {
384                 return @break;
385             }
386             return Each(new Link<TLinkAddress>(index, any, value), handler);
387         }
388         else
389         {
390             if (!Exists(index))
391             {
392                 return @continue;
393             }
394             if (AreEqual(value, any))
395             {
396                 return handler(GetLinkStruct(index));
397             }
398             ref var storedLinkValue = ref GetLinkReference(index);
399             if (AreEqual(storedLinkValue.Source, value) ||
400                 AreEqual(storedLinkValue.Target, value))
401             {
402                 return handler(GetLinkStruct(index));
403             }
404             return @continue;
405         }
406     }

```

```

406     }
407     if (restriction.Count == 3)
408     {
409         var source = restriction[constants.SourcePart];
410         var target = restriction[constants.TargetPart];
411         if (AreEqual(index, any))
412         {
413             if (AreEqual(source, any) && AreEqual(target, any))
414             {
415                 return Each(Array.Empty<TLinkAddress>(), handler);
416             }
417             else if (AreEqual(source, any))
418             {
419                 return TargetsTreeMethods.EachUsage(target, handler);
420             }
421             else if (AreEqual(target, any))
422             {
423                 return SourcesTreeMethods.EachUsage(source, handler);
424             }
425             else //if(source != Any && target != Any)
426             {
427                 var link = SourcesTreeMethods.Search(source, target);
428                 return AreEqual(link, constants.Null) ? @continue :
429                     ↪ handler(GetLinkStruct(link));
430             }
431         }
432         else
433         {
434             if (!Exists(index))
435             {
436                 return @continue;
437             }
438             if (AreEqual(source, any) && AreEqual(target, any))
439             {
440                 return handler(GetLinkStruct(index));
441             }
442             ref var storedLinkValue = ref GetLinkReference(index);
443             if (!AreEqual(source, any) && !AreEqual(target, any))
444             {
445                 if (AreEqual(storedLinkValue.Source, source) &&
446                     AreEqual(storedLinkValue.Target, target))
447                 {
448                     return handler(GetLinkStruct(index));
449                 }
450                 return @continue;
451             }
452             var value = default(TLinkAddress);
453             if (AreEqual(source, any))
454             {
455                 value = target;
456             }
457             if (AreEqual(target, any))
458             {
459                 value = source;
460             }
461             if (AreEqual(storedLinkValue.Source, value) ||
462                 AreEqual(storedLinkValue.Target, value))
463             {
464                 return handler(GetLinkStruct(index));
465             }
466             return @continue;
467         }
468     }
469     throw new NotSupportedException("Другие размеры и способы ограничений не
470     ↪ поддерживаются.");
471 }
472
473 /// <remarks>
474 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
475 ↪ в другом месте (но не в менеджере памяти, а в логике Links)
476 /// </remarks>
477 [MethodImpl(MethodImplOptions.AggressiveInlining)]
478 public virtual TLinkAddress Update(IList<TLinkAddress>? restriction,
479     ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
480 {
481     var constants = Constants;
482     var @null = constants.Null;
483     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
486     // → предварительно заполнено нулями
487     if (!AreEqual(link.Source, @null))
488     {
489         SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
490     }
491     if (!AreEqual(link.Target, @null))
492     {
493         TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
494     }
495     link.Source = substitution[constants.SourcePart];
496     link.Target = substitution[constants.TargetPart];
497     if (!AreEqual(link.Source, @null))
498     {
499         SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
500     }
501     if (!AreEqual(link.Target, @null))
502     {
503         TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
504     }
505     return handler?.Invoke(before, GetLinkStruct(linkIndex)) ?? Constants.Continue;
506 }
507
508 /// <remarks>
509 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
510 /// → пространство
511 /// </remarks>
512 [MethodImpl(MethodImplOptions.AggressiveInlining)]
513 public virtual TLinkAddress Create(ICollection<TLinkAddress>? substitution,
514     // → WriteHandler<TLinkAddress>? handler)
515 {
516     ref var header = ref GetHeaderReference();
517     var freeLink = header.FirstFreeLink;
518     if (!AreEqual(freeLink, Constants.Null))
519     {
520         UnusedLinksListMethods.Detach(freeLink);
521     }
522     else
523     {
524         {
525             var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
526             if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
527             {
528                 throw new
529                     // → LinksLimitReachedException<TLinkAddress>(maximumPossibleInnerReference);
530             }
531             if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
532             {
533                 _memory.ReservedCapacity += _memory.ReservationStep;
534                 SetPointers(_memory);
535                 header = ref GetHeaderReference();
536                 header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
537                     // → LinkSizeInBytes);
538             }
539             freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
540             _memory.UsedCapacity += LinkSizeInBytes;
541         }
542     }
543     return handler?.Invoke(null, new Link<TLinkAddress>(freeLink, Constants.Null,
544         // → Constants.Null)) ?? Constants.Continue;
545 }
546
547 /// <summary>
548 /// <para>
549 /// Deletes the substitution.
550 /// </para>
551 /// <para></para>
552 /// </summary>
553 /// <param name="restriction">
554 /// <para>The substitution.</para>
555 /// <para></para>
556 /// </param>
557 [MethodImpl(MethodImplOptions.AggressiveInlining)]
558 public virtual TLinkAddress Delete(ICollection<TLinkAddress>? restriction,
559     // → WriteHandler<TLinkAddress>? handler)

```

```

551 {
552     ref var header = ref GetHeaderReference();
553     var link = restriction[Constants.IndexPart];
554     var before = GetLinkStruct(link);
555     if (LessThan(link, header.AllocatedLinks))
556     {
557         UnusedLinksListMethods.AttachAsFirst(link);
558         return handler?.Invoke(before, null) ?? Constants.Continue;
559     }
560     else if (AreEqual(link, header.AllocatedLinks))
561     {
562         header.AllocatedLinks = Decrement(header.AllocatedLinks);
563         _memory.UsedCapacity -= LinkSizeInBytes;
564         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
565         // → пока не дойдём до первой существующей связи
566         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
567         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
568             → IsUnusedLink(header.AllocatedLinks))
569         {
570             UnusedLinksListMethods.Detach(header.AllocatedLinks);
571             header.AllocatedLinks = Decrement(header.AllocatedLinks);
572             _memory.UsedCapacity -= LinkSizeInBytes;
573         }
574         return handler?.Invoke(before, null) ?? Constants.Continue;
575     }
576     return Constants.Continue;
577 }
578
579 /// <summary>
580 /// <para>
581 /// Gets the link struct using the specified link index.
582 /// </para>
583 /// <para></para>
584 /// </summary>
585 /// <param name="linkIndex">
586 /// <para>The link index.</para>
587 /// <para></para>
588 /// </param>
589 /// <returns>
590 /// <para>A list of t link</para>
591 /// <para></para>
592 /// </returns>
593 [MethodImpl(MethodImplOptions.AggressiveInlining)]
594 public IList<TLinkAddress>? GetLinkStruct(TLinkAddress linkIndex)
595 {
596     ref var link = ref GetLinkReference(linkIndex);
597     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
598 }
599
600 /// <remarks>
601 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
602 // → адрес реально поменялся
603 ///
604 /// Указатель this.links может быть в том же месте,
605 /// так как 0-я связь не используется и имеет такой же размер как Header,
606 /// поэтому header размещается в том же месте, что и 0-я связь
607 /// </remarks>
608 [MethodImpl(MethodImplOptions.AggressiveInlining)]
609 protected abstract void SetPointers(IResizableDirectMemory memory);
610
611 /// <summary>
612 /// <para>
613 /// Resets the pointers.
614 /// </para>
615 /// <para></para>
616 /// </summary>
617 [MethodImpl(MethodImplOptions.AggressiveInlining)]
618 protected virtual void ResetPointers()
619 {
620     SourcesTreeMethods = null;
621     TargetsTreeMethods = null;
622     UnusedLinksListMethods = null;
623 }
624
625 /// <summary>
626 /// <para>
627 /// Gets the header reference.
628 /// </para>

```

```

626     /// <para></para>
627     /// </summary>
628     /// <returns>
629     /// <para>A ref links header of t link</para>
630     /// <para></para>
631     /// </returns>
632     [MethodImpl(MethodImplOptions.AggressiveInlining)]
633     protected abstract ref LinksHeader<TLinkAddress> GetHeaderReference();
634
635     /// <summary>
636     /// <para>
637     /// Gets the link reference using the specified link index.
638     /// </para>
639     /// <para></para>
640     /// </summary>
641     /// <param name="linkIndex">
642     /// <para>The link index.</para>
643     /// <para></para>
644     /// </param>
645     /// <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<TLinkAddress> GetLinkReference(TLinkAddress 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(TLinkAddress 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(TLinkAddress 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.

```

```

703     /// </para>
704     /// <para></para>
705     /// </summary>
706     /// <returns>
707     /// <para>The link</para>
708     /// <para></para>
709     /// </returns>
710     [MethodImpl(MethodImplOptions.AggressiveInlining)]
711     protected virtual TLinkAddress GetOne() => _one;
712
713     /// <summary>
714     /// <para>
715     /// Gets the zero.
716     /// </para>
717     /// <para></para>
718     /// </summary>
719     /// <returns>
720     /// <para>The link</para>
721     /// <para></para>
722     /// </returns>
723     [MethodImpl(MethodImplOptions.AggressiveInlining)]
724     protected virtual TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress second) =>
768         ↪ _comparer.Compare(first, 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">

```

```

779    /// <para>The second.</para>
780    /// <para></para>
781    /// </param>
782    /// <returns>
783    /// <para>The bool</para>
784    /// <para></para>
785    /// </returns>
786    [MethodImpl(MethodImplOptions.AggressiveInlining)]
787    protected virtual bool LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
788        ↪ _comparer.Compare(first, second) <= 0;
789
790    /// <summary>
791    /// <para>
792    /// Determines whether this instance greater than.
793    /// </para>
794    /// <para></para>
795    /// </summary>
796    /// <param name="first">
797    /// <para>The first.</para>
798    /// <para></para>
799    /// </param>
800    /// <param name="second">
801    /// <para>The second.</para>
802    /// <para></para>
803    /// </param>
804    /// <returns>
805    /// <para>The bool</para>
806    /// <para></para>
807    /// </returns>
808    [MethodImpl(MethodImplOptions.AggressiveInlining)]
809    protected virtual bool GreaterThan(TLinkAddress first, TLinkAddress second) =>
810        ↪ _comparer.Compare(first, second) > 0;
811
812    /// <summary>
813    /// <para>
814    /// Determines whether this instance greater or equal than.
815    /// </para>
816    /// <para></para>
817    /// </summary>
818    /// <param name="first">
819    /// <para>The first.</para>
820    /// <para></para>
821    /// </param>
822    /// <param name="second">
823    /// <para>The second.</para>
824    /// <para></para>
825    /// </param>
826    /// <returns>
827    /// <para>The bool</para>
828    /// <para></para>
829    /// </returns>
830    [MethodImpl(MethodImplOptions.AggressiveInlining)]
831    protected virtual bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
832        ↪ _comparer.Compare(first, second) >= 0;
833
834    /// <summary>
835    /// <para>
836    /// Converts the to int 64 using the specified value.
837    /// </para>
838    /// <para></para>
839    /// </summary>
840    /// <param name="value">
841    /// <para>The value.</para>
842    /// <para></para>
843    /// </param>
844    /// <returns>
845    /// <para>The long</para>
846    /// <para></para>
847    /// </returns>
848    [MethodImpl(MethodImplOptions.AggressiveInlining)]
849    protected virtual long ConvertToInt64(TLinkAddress value) =>
850        ↪ _addressToInt64Converter.Convert(value);
851
852    /// <summary>
853    /// <para>
854    /// Converts the to address using the specified value.
855    /// </para>
856    /// <para></para>

```

```

853     /// </summary>
854     /// <param name="value">
855     /// <para>The value.</para>
856     /// <para></para>
857     /// </param>
858     /// <returns>
859     /// <para>The link</para>
860     /// <para></para>
861     /// </returns>
862     [MethodImpl(MethodImplOptions.AggressiveInlining)]
863     protected virtual TLinkAddress ConvertToAddress(long value) =>
864         ↪ _int64ToAddressConverter.Convert(value);
865
866     /// <summary>
867     /// <para>
868     /// Adds the first.
869     /// </para>
870     /// <para></para>
871     /// </summary>
872     /// <param name="first">
873     /// <para>The first.</para>
874     /// <para></para>
875     /// </param>
876     /// <param name="second">
877     /// <para>The second.</para>
878     /// <para></para>
879     /// </param>
880     /// <returns>
881     /// <para>The link</para>
882     /// <para></para>
883     /// </returns>
884     [MethodImpl(MethodImplOptions.AggressiveInlining)]
885     protected virtual TLinkAddress Add(TLinkAddress first, TLinkAddress second) =>
886         ↪ Arithmetic<TLinkAddress>.Add(first, second);
887
888     /// <summary>
889     /// <para>
890     /// Subtracts the first.
891     /// </para>
892     /// <para></para>
893     /// </summary>
894     /// <param name="first">
895     /// <para>The first.</para>
896     /// <para></para>
897     /// </param>
898     /// <param name="second">
899     /// <para>The second.</para>
900     /// <para></para>
901     /// </param>
902     /// <returns>
903     /// <para>The link</para>
904     /// <para></para>
905     /// </returns>
906     [MethodImpl(MethodImplOptions.AggressiveInlining)]
907     protected virtual TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
908         ↪ Arithmetic<TLinkAddress>.Subtract(first, second);
909
910     /// <summary>
911     /// <para>
912     /// Increments the link.
913     /// </para>
914     /// <para></para>
915     /// </summary>
916     /// <param name="link">
917     /// <para>The link.</para>
918     /// <para></para>
919     /// </param>
920     /// <returns>
921     /// <para>The link</para>
922     /// <para></para>
923     /// </returns>
924     [MethodImpl(MethodImplOptions.AggressiveInlining)]
925     protected virtual TLinkAddress Increment(TLinkAddress link) =>
926         ↪ Arithmetic<TLinkAddress>.Increment(link);
927
928     /// <summary>
929     /// <para>
930     /// Decrements the link.

```

```

927     /// </para>
928     /// <para></para>
929     /// </summary>
930     /// <param name="link">
931     /// <para>The link.</para>
932     /// <para></para>
933     /// </param>
934     /// <returns>
935     /// <para>The link</para>
936     /// <para></para>
937     /// </returns>
938     [MethodImpl(MethodImplOptions.AggressiveInlining)]
939     protected virtual TLinkAddress Decrement(TLinkAddress link) =>
        ↪ Arithmetic<TLinkAddress>.Decrement(link);

940
941     #region Disposable
942
943     /// <summary>
944     /// <para>
945     /// Gets the allow multiple dispose calls value.
946     /// </para>
947     /// <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{TLinkAddress}"/>
17     /// <seealso cref="ILinksListMethods{TLinkAddress}"/>
18     public unsafe class UnusedLinksListMethods<TLinkAddress> :
        ↪ AbsoluteCircularDoublyLinkedListMethods<TLinkAddress>, ILinksListMethods<TLinkAddress>
19     {

```

```

20 private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
    ↪ = UncheckedConverter<TLinkAddress, long>.Default;
21 private readonly byte* _links;
22 private readonly byte* _header;
23
24 /// <summary>
25 /// <para>
26 /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
27 /// </para>
28 /// <para></para>
29 /// </summary>
30 /// <param name="links">
31 /// <para>A links.</para>
32 /// <para></para>
33 /// </param>
34 /// <param name="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<TLinkAddress> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLinkAddress>>(_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<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
    ↪ AsRef<RawLink<TLinkAddress>>(_links + (RawLink<TLinkAddress>.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 TLinkAddress 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 TLinkAddress 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 TLinkAddress GetPrevious(TLinkAddress element) =>
117         ↪ GetLinkReference(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 TLinkAddress GetNext(TLinkAddress element) =>
135         ↪ GetLinkReference(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 TLinkAddress 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(TLinkAddress element) =>
162         ↪ GetHeaderReference().FirstFreeLink = 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>

```

```

169     /// <para></para>
170     /// </param>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     protected override void SetLast(TLinkAddress element) =>
173         ↪ GetHeaderReference().LastFreeLink = element;
174
175     /// <summary>
176     /// <para>
177     /// Sets the previous using the specified element.
178     /// </para>
179     /// </summary>
180     /// <param name="element">
181     /// <para>The element.</para>
182     /// </para>
183     /// </param>
184     /// <param name="previous">
185     /// <para>The previous.</para>
186     /// </para>
187     /// </param>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override void SetPrevious(TLinkAddress element, TLinkAddress previous) =>
190         ↪ GetLinkReference(element).Source = previous;
191
192     /// <summary>
193     /// <para>
194     /// Sets the next using the specified element.
195     /// </para>
196     /// </summary>
197     /// <param name="element">
198     /// <para>The element.</para>
199     /// </para>
200     /// </param>
201     /// <param name="next">
202     /// <para>The next.</para>
203     /// </para>
204     /// </param>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override void SetNext(TLinkAddress element, TLinkAddress next) =>
207         ↪ GetLinkReference(element).Target = next;
208
209     /// <summary>
210     /// <para>
211     /// Sets the size using the specified size.
212     /// </para>
213     /// </summary>
214     /// <param name="size">
215     /// <para>The size.</para>
216     /// </para>
217     /// </param>
218     [MethodImpl(MethodImplOptions.AggressiveInlining)]
219     protected override void SetSize(TLinkAddress size) => GetHeaderReference().FreeLinks =
220         ↪ 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     /// </summary>
15     public struct RawLink<TLinkAddress> : IEquatable<RawLink<TLinkAddress>>
16     {
17         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
18             ↪ EqualityComparer<TLinkAddress>.Default;
19

```

```

20     /// <summary>
21     /// <para>
22     /// The size.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     public static readonly long SizeInBytes = Structure<RawLink<TLinkAddress>>.Size;
27
28     /// <summary>
29     /// <para>
30     /// The source.
31     /// </para>
32     /// <para></para>
33     /// </summary>
34     public TLinkAddress Source;
35     /// <summary>
36     /// <para>
37     /// The target.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public TLinkAddress Target;
42     /// <summary>
43     /// <para>
44     /// The left as source.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     public TLinkAddress LeftAsSource;
49     /// <summary>
50     /// <para>
51     /// The right as source.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     public TLinkAddress RightAsSource;
56     /// <summary>
57     /// <para>
58     /// The size as source.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLinkAddress SizeAsSource;
63     /// <summary>
64     /// <para>
65     /// The left as target.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public TLinkAddress LeftAsTarget;
70     /// <summary>
71     /// <para>
72     /// The right as target.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public TLinkAddress RightAsTarget;
77     /// <summary>
78     /// <para>
79     /// The size as target.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLinkAddress 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<TLinkAddress> 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<TLinkAddress> other)
118        => _equalityComparer.Equals(Source, other.Source)
119        && _equalityComparer.Equals(Target, other.Target)
120        && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
121        && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
122        && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
123        && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
124        && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
125        && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);

126
127    /// <summary>
128    /// <para>
129    /// Gets the hash code.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <returns>
134    /// <para>The int</para>
135    /// <para></para>
136    /// </returns>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    public override int GetHashCode() => (Source, Target, LeftAsSource, RightAsSource,
        ↳ SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();

139
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    public static bool operator ==(RawLink<TLinkAddress> left, RawLink<TLinkAddress> right)
142        ↳ => left.Equals(right);

143
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    public static bool operator !=(RawLink<TLinkAddress> left, RawLink<TLinkAddress> right)
146        ↳ => !(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 :
        ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<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="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
35     ↪ 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 UInt32LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<uint>
52     ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header)
53     : base(constants, (byte*)links, (byte*)header)
54     {
55         Links = links;
56         Header = header;
57     }
58
59     /// <summary>
60     /// <para>
61     /// Gets the zero.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <returns>
66     /// <para>The uint</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override uint GetZero() => 0U;
71
72     /// <summary>
73     /// <para>
74     /// Determines whether this instance equal to zero.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="value">
79     /// <para>The value.</para>
80     /// <para></para>
81     /// </param>
82     /// <returns>
83     /// <para>The bool</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected override bool EqualToZero(uint value) => value == 0U;
88
89     /// <summary>
90     /// <para>
91     /// Determines whether this instance are equal.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="first">
96     /// <para>The first.</para>
97     /// <para></para>
98     /// </param>
99     /// <param name="second">
100    /// <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
    ↪ 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
    ↪ always >= 0 for uint

200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance less or equal than.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="first">
208     /// <para>The first.</para>
209     /// <para></para>
210     /// </param>
211     /// <param name="second">
212     /// <para>The second.</para>
213     /// <para></para>
214     /// </param>
215     /// <returns>
216     /// <para>The bool</para>
217     /// <para></para>
218     /// </returns>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override bool LessOrEqualThan(uint first, uint second) => first <= second;

221
222     /// <summary>
223     /// <para>
224     /// Determines whether this instance less than zero.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <param name="value">
229     /// <para>The value.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool LessThanZero(uint value) => false; // value < 0 is always false
    ↪ for uint

238
239     /// <summary>
240     /// <para>
241     /// Determines whether this instance less than.
242     /// </para>
243     /// <para></para>
244     /// </summary>
245     /// <param name="first">
246     /// <para>The first.</para>
247     /// <para></para>
248     /// </param>
249     /// <param name="second">
250     /// <para>The second.</para>
251     /// <para></para>

```

```

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

```



```

330 /// <para>The uint</para>
331 /// <para></para>
332 /// </returns>
333 [MethodImpl(MethodImplOptions.AggressiveInlining)]
334 protected override uint Subtract(uint first, uint second) => first - second;
335
336 /// <summary>
337 /// <para>
338 /// Determines whether this instance first is to the left of second.
339 /// </para>
340 /// <para></para>
341 /// </summary>
342 /// <param name="first">
343 /// <para>The first.</para>
344 /// <para></para>
345 /// </param>
346 /// <param name="second">
347 /// <para>The second.</para>
348 /// <para></para>
349 /// </param>
350 /// <returns>
351 /// <para>The bool</para>
352 /// <para></para>
353 /// </returns>
354 [MethodImpl(MethodImplOptions.AggressiveInlining)]
355 protected override bool FirstIsToLeftOfSecond(uint first, uint second)
356 {
357     ref var firstLink = ref Links[first];
358     ref var secondLink = ref Links[second];
359     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
360         ↪ secondLink.Source, secondLink.Target);
361 }
362
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>
397 /// <para>A ref links header of uint</para>
398 /// <para></para>
399 /// </returns>
400 [MethodImpl(MethodImplOptions.AggressiveInlining)]
401 protected override ref LinksHeader<uint> GetHeaderReference() => ref *Header;
402
403 /// <summary>
404 /// <para>
405 /// Gets the link reference using the specified link.
406 /// </para>
407 /// <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 :
16     ↪ LinksSizeBalancedTreeMethodsBase<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="UInt32LinksSizeBalancedTreeMethodsBase"/> instance.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="constants">
41         /// <para>A constants.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="links">
45         /// <para>A links.</para>
46         /// <para></para>
47         /// </param>
48         /// <param name="header">
49         /// <para>A header.</para>
50         /// <para></para>
51         /// </param>
52         protected UInt32LinksSizeBalancedTreeMethodsBase(LinksConstants<uint> constants,
53     ↪ RawLink<uint>* links, LinksHeader<uint>* header)
54         : base(constants, (byte*)links, (byte*)header)
55         {
56             Links = links;
57             Header = header;
58         }
59
60         /// <summary>
61         /// <para>
62         /// Gets the zero.
63         /// </para>
64         /// <para></para>
65         /// </summary>

```

```

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
    ↪ 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
    ↪ always >= 0 for uint
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance less or equal than.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="first">
208     /// <para>The first.</para>
209     /// <para></para>
210     /// </param>
211     /// <param name="second">
212     /// <para>The second.</para>
213     /// <para></para>
214     /// </param>
215     /// <returns>
216     /// <para>The bool</para>

```

```

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

```

```

294    /// <summary>
295    /// <para>
296    /// Adds the first.
297    /// </para>
298    /// <para></para>
299    /// </summary>
300    /// <param name="first">
301    /// <para>The first.</para>
302    /// <para></para>
303    /// </param>
304    /// <param name="second">
305    /// <para>The second.</para>
306    /// <para></para>
307    /// </param>
308    /// <returns>
309    /// <para>The uint</para>
310    /// <para></para>
311    /// </returns>
312    [MethodImpl(MethodImplOptions.AggressiveInlining)]
313    protected override uint Add(uint first, uint second) => first + second;
314
315    /// <summary>
316    /// <para>
317    /// Subtracts the first.
318    /// </para>
319    /// <para></para>
320    /// </summary>
321    /// <param name="first">
322    /// <para>The first.</para>
323    /// <para></para>
324    /// </param>
325    /// <param name="second">
326    /// <para>The second.</para>
327    /// <para></para>
328    /// </param>
329    /// <returns>
330    /// <para>The uint</para>
331    /// <para></para>
332    /// </returns>
333    [MethodImpl(MethodImplOptions.AggressiveInlining)]
334    protected override uint Subtract(uint first, uint second) => first - second;
335
336    /// <summary>
337    /// <para>
338    /// Determines whether this instance first is to the left of second.
339    /// </para>
340    /// <para></para>
341    /// </summary>
342    /// <param name="first">
343    /// <para>The first.</para>
344    /// <para></para>
345    /// </param>
346    /// <param name="second">
347    /// <para>The second.</para>
348    /// <para></para>
349    /// </param>
350    /// <returns>
351    /// <para>The bool</para>
352    /// <para></para>
353    /// </returns>
354    [MethodImpl(MethodImplOptions.AggressiveInlining)]
355    protected override bool FirstIsToLeftOfSecond(uint first, uint second)
356    {
357        ref var firstLink = ref Links[first];
358        ref var secondLink = ref Links[second];
359        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
360            ↪ secondLink.Source, secondLink.Target);
361    }
362
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(uint first, uint second)
382     {
383         ref var firstLink = ref Links[first];
384         ref var secondLink = ref Links[second];
385         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
386             ↪ secondLink.Source, secondLink.Target);
387     }
388     /// <summary>
389     /// <para>
390     /// Gets the header reference.
391     /// </para>
392     /// <para></para>
393     /// </summary>
394     /// <returns>
395     /// <para>A ref links header of uint</para>
396     /// <para></para>
397     /// </returns>
398     [MethodImpl(MethodImplOptions.AggressiveInlining)]
399     protected override ref LinksHeader<uint> GetHeaderReference() => ref *Header;
400
401     /// <summary>
402     /// <para>
403     /// Gets the link reference using the specified link.
404     /// </para>
405     /// <para></para>
406     /// </summary>
407     /// <param name="link">
408     /// <para>The link.</para>
409     /// <para></para>
410     /// </param>
411     /// <returns>
412     /// <para>A ref raw link of uint</para>
413     /// <para></para>
414     /// </returns>
415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     protected override ref RawLink<uint> GetLinkReference(uint link) => ref Links[link];
417 }
418 }

```

1.95 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTree

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods :
15        ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↪ cref="UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>

```

```

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 UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(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].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
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>

```

```

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,
260     ↪ uint secondSource, uint secondTarget)
261     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
262     ↪ secondTarget);
263
264     /// <summary>
265     /// <para>
266     /// Clears the node using the specified node.
267     /// </para>
268     /// <para></para>
269     /// </summary>
270     /// <param name="node">
271     /// <para>The node.</para>
272     /// <para></para>
273     /// </param>
274     [MethodImpl(MethodImplOptions.AggressiveInlining)]
275     protected override void ClearNode(uint node)
276     {
277         ref var link = ref Links[node];
278         link.LeftAsSource = 0U;
279         link.RightAsSource = 0U;
280         link.SizeAsSource = 0U;
281     }
282 }

```

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         /// <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 UInt32LinksSourcesSizeBalancedTreeMethods(LinksConstants<uint> constants,
36         ↪ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
37
38         /// <summary>
39         /// <para>
40         /// Gets the left reference using the specified node.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         /// <param name="node">
45         /// <para>The node.</para>
46         /// <para></para>

```

```

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].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
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>

```

```

272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void ClearNode(uint node)
274     {
275         ref var link = ref Links[node];
276         link.LeftAsSource = 0U;
277         link.RightAsSource = 0U;
278         link.SizeAsSource = 0U;
279     }
280 }
281 }

```

1.97 ./csharp/Platform.Data.Doublets/Memory/United/Specific/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 :
15         ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
37             ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
38             ↪ header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="node">
47         /// <para>The node.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The ref uint</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref uint GetLeftReference(uint node) => ref Links[node].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>

```

```

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

```



```

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

```

```

217     /// <para>The second source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToTheLeftOfSecond(uint firstSource, uint firstTarget,
    ↪ uint secondSource, uint secondTarget)
    ↪ => 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(uint firstSource, uint firstTarget,
    ↪ uint secondSource, uint secondTarget)
    ↪ => 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 :
15     ↳ UInt32LinksSizeBalancedTreeMethodsBase
16 {
17     /// <summary>
18     /// <para>
19     /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> instance.
20     /// </para>
21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// </param>
25     /// <param name="links">
26     /// <para>A links.</para>
27     /// </param>
28     /// <param name="header">
29     /// <para>A header.</para>
30     /// </param>
31     public UInt32LinksTargetsSizeBalancedTreeMethods(LinksConstants<uint> constants,
32     ↳ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
33
34     /// <summary>
35     /// <para>
36     /// Gets the left reference using the specified node.
37     /// </para>
38     /// </summary>
39     /// <param name="node">
40     /// <para>The node.</para>
41     /// </param>
42     /// <returns>
43     /// <para>The ref uint</para>
44     /// </returns>
45     [MethodImpl(MethodImplOptions.AggressiveInlining)]
46     protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsTarget;
47
48     /// <summary>
49     /// <para>
50     /// Gets the right reference using the specified node.
51     /// </para>
52     /// </summary>
53     /// <param name="node">
54     /// <para>The node.</para>
55     /// </param>
56     /// <returns>
57     /// <para>The ref uint</para>
58     /// </returns>
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     protected override ref uint GetRightReference(uint node) => ref
61     ↳ Links[node].RightAsTarget;
62
63     /// <summary>
64     /// <para>
65     /// Gets the left using the specified node.
66     /// </para>
67     /// </summary>
68     /// <param name="node">
69     /// <para>The node.</para>
70     /// </param>
71     /// <returns>
72     /// <para>The uint</para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected override uint GetLeftReference(uint node) => Links[node].LeftAsTarget;
76
77     /// <summary>
78     /// <para>
79     /// Gets the right using the specified node.
80     /// </para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// </param>
85     /// <returns>
86     /// <para>The uint</para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override uint GetRightReference(uint node) => Links[node].RightAsTarget;

```

```

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

```

```

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

```

```

236     /// <para></para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="firstTarget">
243     /// <para>The first target.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
    ↪ uint secondSource, uint secondTarget)
260     => 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
    ↪ organizing the storage of links with addresses represented as <see cref="uint" />.</para>
13     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
    ↪ размером, для организации хранения связей с адресами представленными в виде <see
    ↪ cref="uint"/>.</para>
14     /// </summary>
15     public unsafe class UInt32UnitedMemoryLinks : UnitedMemoryLinksBase<uint>
16     {
17         private readonly Func<ILinksTreeMethods<uint>> _createSourceTreeMethods;
18         private readonly Func<ILinksTreeMethods<uint>> _createTargetTreeMethods;
19         private LinksHeader<uint>* _header;
20         private RawLink<uint>* _links;
21
22         /// <summary>
23         /// <para>
24         /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> 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 UInt32UnitedMemoryLinks(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 UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
44     ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
45     ↪ memoryReservationStep) { }
46
47     /// <summary>
48     /// <para>
49     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> 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 UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
59     ↪ DefaultLinksSizeStep) { }
60
61     /// <summary>
62     /// <para>
63     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> 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 UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
77     ↪ memoryReservationStep) : this(memory, memoryReservationStep,
78     ↪ Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }
79
80     /// <summary>
81     /// <para>
82     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> 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>
98     /// <param name="indexTreeType">
99     /// <para>A index tree type.</para>
100    /// <para></para>
101    /// </param>
102    [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>
192     /// <para></para>
193     /// </returns>
194     [MethodImpl(MethodImplOptions.AggressiveInlining)]
195     protected override bool AreEqual(uint first, uint second) => first == second;
196
197     /// <summary>
198     /// <para>
199     /// Determines whether this instance less than.
200     /// </para>
201     /// <para></para>
202     /// </summary>
203     /// <param name="first">
204     /// <para>The first.</para>
205     /// <para></para>
206     /// </param>
207     /// <param name="second">
208     /// <para>The second.</para>
209     /// <para></para>
210     /// </param>
211     /// <returns>
212     /// <para>The bool</para>
213     /// <para></para>
214     /// </returns>
215     [MethodImpl(MethodImplOptions.AggressiveInlining)]
216     protected override bool LessThan(uint first, uint second) => first < second;
217
218     /// <summary>
219     /// <para>
220     /// Determines whether this instance less or equal than.
221     /// </para>
222     /// <para></para>
223     /// </summary>
224     /// <param name="first">
225     /// <para>The first.</para>
226     /// <para></para>
227     /// </param>
228     /// <param name="second">
229     /// <para>The second.</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 LessOrEqualThan(uint first, uint second) => first <= second;
238
239     /// <summary>
240     /// <para>
241     /// Determines whether this instance greater than.
242     /// </para>
243     /// <para></para>

```

```

243     /// </summary>
244     /// <param name="first">
245     /// <para>The first.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="second">
249     /// <para>The second.</para>
250     /// <para></para>
251     /// </param>
252     /// <returns>
253     /// <para>The bool</para>
254     /// <para></para>
255     /// </returns>
256     [MethodImpl(MethodImplOptions.AggressiveInlining)]
257     protected override bool GreaterThan(uint first, uint second) => first > second;
258
259     /// <summary>
260     /// <para>
261     /// Determines whether this instance greater or equal than.
262     /// </para>
263     /// <para></para>
264     /// </summary>
265     /// <param name="first">
266     /// <para>The first.</para>
267     /// <para></para>
268     /// </param>
269     /// <param name="second">
270     /// <para>The second.</para>
271     /// <para></para>
272     /// </param>
273     /// <returns>
274     /// <para>The bool</para>
275     /// <para></para>
276     /// </returns>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
279
280     /// <summary>
281     /// <para>
282     /// Gets the zero.
283     /// </para>
284     /// <para></para>
285     /// </summary>
286     /// <returns>
287     /// <para>The uint</para>
288     /// <para></para>
289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     protected override uint GetZero() => 0U;
292
293     /// <summary>
294     /// <para>
295     /// Gets the one.
296     /// </para>
297     /// <para></para>
298     /// </summary>
299     /// <returns>
300     /// <para>The uint</para>
301     /// <para></para>
302     /// </returns>
303     [MethodImpl(MethodImplOptions.AggressiveInlining)]
304     protected override uint GetOne() => 1U;
305
306     /// <summary>
307     /// <para>
308     /// Converts the to int 64 using the specified value.
309     /// </para>
310     /// <para></para>
311     /// </summary>
312     /// <param name="value">
313     /// <para>The value.</para>
314     /// <para></para>
315     /// </param>
316     /// <returns>
317     /// <para>The long</para>
318     /// <para></para>
319     /// </returns>
320     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

321     protected override long ConvertToInt64(uint value) => (long)value;
322
323     /// <summary>
324     /// <para>
325     /// Converts the to address using the specified value.
326     /// </para>
327     /// <para></para>
328     /// </summary>
329     /// <param name="value">
330     /// <para>The value.</para>
331     /// <para></para>
332     /// </param>
333     /// <returns>
334     /// <para>The uint</para>
335     /// <para></para>
336     /// </returns>
337     [MethodImpl(MethodImplOptions.AggressiveInlining)]
338     protected override uint ConvertToAddress(long value) => (uint)value;
339
340     /// <summary>
341     /// <para>
342     /// Adds the first.
343     /// </para>
344     /// <para></para>
345     /// </summary>
346     /// <param name="first">
347     /// <para>The first.</para>
348     /// <para></para>
349     /// </param>
350     /// <param name="second">
351     /// <para>The second.</para>
352     /// <para></para>
353     /// </param>
354     /// <returns>
355     /// <para>The uint</para>
356     /// <para></para>
357     /// </returns>
358     [MethodImpl(MethodImplOptions.AggressiveInlining)]
359     protected override uint Add(uint first, uint second) => first + second;
360
361     /// <summary>
362     /// <para>
363     /// Subtracts the first.
364     /// </para>
365     /// <para></para>
366     /// </summary>
367     /// <param name="first">
368     /// <para>The first.</para>
369     /// <para></para>
370     /// </param>
371     /// <param name="second">
372     /// <para>The second.</para>
373     /// <para></para>
374     /// </param>
375     /// <returns>
376     /// <para>The uint</para>
377     /// <para></para>
378     /// </returns>
379     [MethodImpl(MethodImplOptions.AggressiveInlining)]
380     protected override uint Subtract(uint first, uint second) => first - second;
381
382     /// <summary>
383     /// <para>
384     /// Increments the link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>The uint</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override uint Increment(uint link) => ++link;
398

```

```

399     /// <summary>
400     /// <para>
401     /// Decrements the link.
402     /// </para>
403     /// <para></para>
404     /// </summary>
405     /// <param name="link">
406     /// <para>The link.</para>
407     /// <para></para>
408     /// </param>
409     /// <returns>
410     /// <para>The uint</para>
411     /// <para></para>
412     /// </returns>
413     [MethodImpl(MethodImplOptions.AggressiveInlining)]
414     protected override uint Decrement(uint link) => --link;
415 }
416 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 unused links list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UnusedLinksListMethods{uint}"/>
15     public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<uint>
16     {
17         private readonly RawLink<uint>* _links;
18         private readonly LinksHeader<uint>* _header;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>
32         /// <para></para>
33         /// </param>
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         public UInt32UnusedLinksListMethods(RawLink<uint>* links, LinksHeader<uint>* header)
36             : base((byte*)links, (byte*)header)
37         {
38             _links = links;
39             _header = header;
40         }
41
42         /// <summary>
43         /// <para>
44         /// Gets the link reference using the specified link.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="link">
49         /// <para>The link.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>A ref raw link of uint</para>
54         /// <para></para>
55         /// </returns>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         protected override ref RawLink<uint> GetLinkReference(uint link) => ref _links[link];
58
59         /// <summary>

```

```

60     /// <para>
61     /// Gets the header reference.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <returns>
66     /// <para>A ref links header of uint</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
71 }
72 }
```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3 using static System.Runtime.CompilerServices.Unsafe;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.United.Specific
8 {
9     /// <summary>
10     /// <para>
11     /// Represents the int 64 links avl balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksAvlBalancedTreeMethodsBase{ulong}" />
16     public unsafe abstract class UInt64LinksAvlBalancedTreeMethodsBase :
17         ↳ LinksAvlBalancedTreeMethodsBase<ulong>
18     {
19         /// <summary>
20         /// <para>
21         /// The links.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLink<ulong>* Links;
26         /// <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="UInt64LinksAvlBalancedTreeMethodsBase" /> instance.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="constants">
41         /// <para>A constants.</para>
42         /// </param>
43         /// <param name="links">
44         /// <para>A links.</para>
45         /// <para></para>
46         /// </param>
47         /// <param name="header">
48         /// <para>A header.</para>
49         /// <para></para>
50         /// </param>
51         protected UInt64LinksAvlBalancedTreeMethodsBase(LinksConstants<ulong> constants,
52             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header)
53             : base(constants, (byte*)links, (byte*)header)
54         {
55             Links = links;
56             Header = header;
57         }
58
59         /// <summary>
60         /// <para>
61         /// Gets the zero.
62         /// </para>
63         /// <para></para>
64         /// </summary>

```

```

63     /// </summary>
64     /// <returns>
65     /// <para>The ulong</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ulong GetZero() => OUL;
70
71     /// <summary>
72     /// <para>
73     /// Determines whether this instance equal to zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="value">
78     /// <para>The value.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The bool</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override bool EqualToZero(ulong value) => value == OUL;
87
88     /// <summary>
89     /// <para>
90     /// Determines whether this instance are equal.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="first">
95     /// <para>The first.</para>
96     /// <para></para>
97     /// </param>
98     /// <param name="second">
99     /// <para>The second.</para>
100    /// <para></para>
101    /// </param>
102    /// <returns>
103    /// <para>The bool</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override bool AreEqual(ulong first, ulong second) => first == second;
108
109    /// <summary>
110    /// <para>
111    /// Determines whether this instance greater than zero.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="value">
116    /// <para>The value.</para>
117    /// <para></para>
118    /// </param>
119    /// <returns>
120    /// <para>The bool</para>
121    /// <para></para>
122    /// </returns>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override bool GreaterThanZero(ulong value) => value > OUL;
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
239     ↪ for ulong
240
241     /// <summary>
242     /// <para>
243     /// Determines whether this instance less than.
244     /// </para>
245     /// <para></para>
246     /// </summary>
247     /// <param name="first">
248     /// <para>The first.</para>
249     /// <para></para>
250     /// </param>
251     /// <param name="second">
252     /// <para>The second.</para>
253     /// <para></para>
254     /// </param>
255     /// <returns>
256     /// <para>The bool</para>
257     /// <para></para>
258     /// </returns>
259     [MethodImpl(MethodImplOptions.AggressiveInlining)]
260     protected override bool LessThan(ulong first, ulong second) => first < second;
261
262     /// <summary>
263     /// <para>
264     /// Increments the value.
265     /// </para>
266     /// <para></para>
267     /// </summary>
268     /// <param name="value">
269     /// <para>The value.</para>
270     /// <para></para>
271     /// </param>
272     /// <returns>
273     /// <para>The ulong</para>
274     /// <para></para>
275     /// </returns>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override ulong Increment(ulong value) => ++value;
278
279     /// <summary>
280     /// <para>
281     /// Decrements the 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 ulong</para>
291     /// <para></para>
292     /// </returns>
293     [MethodImpl(MethodImplOptions.AggressiveInlining)]
294     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>
372    /// </param>
373    /// <param name="second">
374    /// <para>The second.</para>
375    /// <para></para>
376    /// </param>
377    /// <returns>
378    /// <para>The bool</para>
379    /// <para></para>
380    /// </returns>
381    [MethodImpl(MethodImplOptions.AggressiveInlining)]
382    protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
383    {
384        ref var firstLink = ref Links[first];
385        ref var secondLink = ref Links[second];
386        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
387            ↪ secondLink.Source, secondLink.Target);
388    }
389    /// <summary>
390    /// <para>
391    /// Gets the size value using the specified value.
392    /// </para>
393    /// <para></para>
394    /// </summary>
395    /// <param name="value">
396    /// <para>The value.</para>
397    /// <para></para>
398    /// </param>
399    /// <returns>
400    /// <para>The ulong</para>
401    /// <para></para>
402    /// </returns>
403    [MethodImpl(MethodImplOptions.AggressiveInlining)]
404    protected override ulong GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;
405
406    /// <summary>
407    /// <para>
408    /// Sets the size value using the specified stored value.
409    /// </para>
410    /// <para></para>
411    /// </summary>
412    /// <param name="storedValue">
413    /// <para>The stored value.</para>
414    /// <para></para>
415    /// </param>
416    /// <param name="size">
417    /// <para>The size.</para>
418    /// <para></para>
419    /// </param>
420    [MethodImpl(MethodImplOptions.AggressiveInlining)]
421    protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
422        ↪ storedValue & 31UL | (size & 134217727UL) << 5;
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">

```

```

447 /// <para>The stored value.</para>
448 /// <para></para>
449 /// </param>
450 /// <param name="value">
451 /// <para>The value.</para>
452 /// <para></para>
453 /// </param>
454 [MethodImpl(MethodImplOptions.AggressiveInlining)]
455 protected override void SetLeftIsChildValue(ref ulong storedValue, bool value) =>
456     ↳ storedValue = storedValue & 4294967279UL | (As<bool, byte>(ref value) & 1UL) << 4;
457
458 /// <summary>
459 /// <para>
460 /// Determines whether this instance get right is child value.
461 /// </para>
462 /// <para></para>
463 /// </summary>
464 /// <param name="value">
465 /// <para>The value.</para>
466 /// <para></para>
467 /// </param>
468 /// <returns>
469 /// <para>The bool</para>
470 /// <para></para>
471 /// </returns>
472 [MethodImpl(MethodImplOptions.AggressiveInlining)]
473 protected override bool GetRightIsChildValue(ulong value) => (value & 8UL) >> 3 == 1UL;
474
475 /// <summary>
476 /// <para>
477 /// Sets the right is child value using the specified stored value.
478 /// </para>
479 /// <para></para>
480 /// </summary>
481 /// <param name="storedValue">
482 /// <para>The stored value.</para>
483 /// <para></para>
484 /// </param>
485 /// <param name="value">
486 /// <para>The value.</para>
487 /// <para></para>
488 /// </param>
489 [MethodImpl(MethodImplOptions.AggressiveInlining)]
490 protected override void SetRightIsChildValue(ref ulong storedValue, bool value) =>
491     ↳ storedValue = storedValue & 4294967287UL | (As<bool, byte>(ref value) & 1UL) << 3;
492
493 /// <summary>
494 /// <para>
495 /// Gets the balance value using the specified value.
496 /// </para>
497 /// <para></para>
498 /// </summary>
499 /// <param name="value">
500 /// <para>The value.</para>
501 /// <para></para>
502 /// </param>
503 /// <returns>
504 /// <para>The sbyte</para>
505 /// <para></para>
506 /// </returns>
507 [MethodImpl(MethodImplOptions.AggressiveInlining)]
508 protected override sbyte GetBalanceValue(ulong value) => unchecked((sbyte)(value & 7UL |
509     ↳ 0xF8UL * ((value & 4UL) >> 2))); // if negative, then continue ones to the end of
510     ↳ sbyte
511
512 /// <summary>
513 /// <para>
514 /// Sets the balance value using the specified stored value.
515 /// </para>
516 /// <para></para>
517 /// </summary>
518 /// <param name="storedValue">
519 /// <para>The stored value.</para>
520 /// <para></para>
521 /// </param>
522 /// <param name="value">
523 /// <para>The value.</para>
524 /// <para></para>
525 /// </param>

```

```

521     /// </param>
522     [MethodImpl(MethodImplOptions.AggressiveInlining)]
523     protected override void SetBalanceValue(ref ulong storedValue, sbyte value) =>
        ↪ storedValue = unchecked(storedValue & 4294967288UL | (ulong)((byte)value >> 5 & 4 |
        ↪ value & 3) & 7UL);
524
525     /// <summary>
526     /// <para>
527     /// Gets the header reference.
528     /// </para>
529     /// <para></para>
530     /// </summary>
531     /// <returns>
532     /// <para>A ref links header of ulong</para>
533     /// <para></para>
534     /// </returns>
535     [MethodImpl(MethodImplOptions.AggressiveInlining)]
536     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
537
538     /// <summary>
539     /// <para>
540     /// Gets the link reference using the specified link.
541     /// </para>
542     /// <para></para>
543     /// </summary>
544     /// <param name="link">
545     /// <para>The link.</para>
546     /// <para></para>
547     /// </param>
548     /// <returns>
549     /// <para>A ref raw link of ulong</para>
550     /// <para></para>
551     /// </returns>
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]
553     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
554 }
555 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 links recursionless size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{ulong}">
15     public unsafe abstract class UInt64LinksRecursionlessSizeBalancedTreeMethodsBase :
        ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<ulong>
16     {
17         /// <summary>
18         /// <para>
19         /// The links.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         protected new readonly RawLink<ulong>* Links;
24         /// <summary>
25         /// <para>
26         /// The header.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         protected new readonly LinksHeader<ulong>* Header;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase">
35         ↪ instance.
36         /// </para>
37         /// <para></para>
38         /// </summary>

```

```

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 UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
    → constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
    : base(constants, (byte*)links, (byte*)header)
51     {
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() => 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 > 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    /// <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(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.
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="UInt64LinksSizeBalancedTreeMethodsBase" /> 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 UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
52    ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
53        : base(constants, (byte*)links, (byte*)header)
54        {
55            Links = links;
56            Header = header;
57        }
58
59        /// <summary>
60        /// <para>
61        /// Gets the zero.
62        /// </para>
63        /// <para></para>
64        /// </summary>
65        /// <returns>
66        /// <para>The ulong</para>
67        /// <para></para>
68        /// </returns>
69        [MethodImpl(MethodImplOptions.AggressiveInlining)]
70        protected override ulong GetZero() => 0UL;
71
72        /// <summary>
73        /// <para>
74        /// Determines whether this instance equal to zero.
75        /// </para>
76        /// <para></para>
77        /// </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     /// <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 > 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];

```

```

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 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 }

```

1.104 ./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 :
        ↪ UInt64LinksAvlBalancedTreeMethodsBase
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="UInt64LinksSourcesAvlBalancedTreeMethods"/> instance.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <param name="constants">
23        /// <para>A constants.</para>
24        /// <para></para>
25        /// </param>
26        /// <param name="links">
27        /// <para>A links.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="header">
31        /// <para>A header.</para>
32        /// <para></para>
33        /// </param>
34        public UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
        ↪ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
        ↪ { }
35
36        /// <summary>
37        /// <para>

```

```

38     /// Gets the left reference using the specified node.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref ulong</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
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     /// <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 // [MethodImpl(MethodImplOptions.AggressiveInlining)]
189 // protected override bool GetLeftIsChild(ulong node) => IsChild(node, GetLeft(node));
190
191 /// <summary>
192 /// <para>
193 /// Sets the left is child using the specified node.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <param name="node">
198 /// <para>The node.</para>
199 /// <para></para>
200 /// </param>
201 /// <param name="value">
202 /// <para>The value.</para>
203 /// <para></para>
204 /// </param>
205 [MethodImpl(MethodImplOptions.AggressiveInlining)]
206 protected override void SetLeftIsChild(ulong node, bool value) =>
207     ↪ SetLeftIsChildValue(ref Links[node].SizeAsSource, value);
208
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);

```

```

262     /// <summary>
263     /// <para>
264     /// Sets the balance 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     /// <param name="value">
273     /// <para>The value.</para>
274     /// <para></para>
275     /// </param>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
278     ↪ Links[node].SizeAsSource, value);
279
280     /// <summary>
281     /// <para>
282     /// Gets the tree root.
283     /// </para>
284     /// <para></para>
285     /// </summary>
286     /// <returns>
287     /// <para>The ulong</para>
288     /// <para></para>
289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     protected override ulong GetTreeRoot() => Header->RootAsSource;
292
293     /// <summary>
294     /// <para>
295     /// Gets the base part value using the specified link.
296     /// </para>
297     /// <para></para>
298     /// </summary>
299     /// <param name="link">
300     /// <para>The link.</para>
301     /// <para></para>
302     /// </param>
303     /// <returns>
304     /// <para>The ulong</para>
305     /// <para></para>
306     /// </returns>
307     [MethodImpl(MethodImplOptions.AggressiveInlining)]
308     protected override ulong GetBasePartValue(ulong link) => Links[link].Source;
309
310     /// <summary>
311     /// <para>
312     /// Determines whether this instance first is to the left of second.
313     /// </para>
314     /// <para></para>
315     /// </summary>
316     /// <param name="firstSource">
317     /// <para>The first source.</para>
318     /// <para></para>
319     /// </param>
320     /// <param name="firstTarget">
321     /// <para>The first target.</para>
322     /// <para></para>
323     /// </param>
324     /// <param name="secondSource">
325     /// <para>The second source.</para>
326     /// <para></para>
327     /// </param>
328     /// <param name="secondTarget">
329     /// <para>The second target.</para>
330     /// <para></para>
331     /// </param>
332     /// <returns>
333     /// <para>The bool</para>
334     /// <para></para>
335     /// </returns>
336     [MethodImpl(MethodImplOptions.AggressiveInlining)]
337     protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
    ↪ ulong secondSource, ulong secondTarget)

```

```

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

```

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 :
        ↪ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see
        ↪ cref="UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
    ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
    ↪ links, header) { }
35
36    /// <summary>
37    /// <para>
38    /// Gets the left reference using the specified node.
39    /// </para>
40    /// <para></para>
41    /// </summary>
42    /// <param name="node">
43    /// <para>The node.</para>
44    /// <para></para>
45    /// </param>
46    /// <returns>
47    /// <para>The ref ulong</para>
48    /// <para></para>
49    /// </returns>
50    [MethodImpl(MethodImplOptions.AggressiveInlining)]
51    protected override ref ulong GetLeftReference(ulong node) => ref
    ↪ Links[node].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 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 =
    ↪ right;
137
138   /// <summary>
139   /// <para>
140   /// Gets the size using the specified node.
141   /// </para>
142   /// <para></para>
143   /// </summary>
144   /// <param name="node">
145   /// <para>The node.</para>
146   /// <para></para>
147   /// </param>
148   /// <returns>
149   /// <para>The ulong</para>
150   /// <para></para>
151   /// </returns>
152   [MethodImpl(MethodImplOptions.AggressiveInlining)]
153   protected override ulong GetSize(ulong node) => Links[node].SizeAsSource;
154
155   /// <summary>
156   /// <para>
157   /// Sets the size using the specified node.
158   /// </para>
159   /// <para></para>
160   /// </summary>
161   /// <param name="node">
162   /// <para>The node.</para>
163   /// <para></para>
164   /// </param>
165   /// <param name="size">
166   /// <para>The size.</para>
167   /// <para></para>
168   /// </param>
169   [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

```

/// </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>
/// <para></para>
/// </param>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override void ClearNode(ulong node)
{
    ref var link = ref Links[node];
    link.LeftAsSource = OUL;
    link.RightAsSource = OUL;
    link.SizeAsSource = OUL;
}

```

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></para>
21        /// </summary>
22        /// <param name="constants">
23        /// <para>A constants.</para>
24        /// <para></para>
25        /// </param>
26        /// <param name="links">
27        /// <para>A links.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="header">
31        /// <para>A header.</para>
32        /// <para></para>
33        /// </param>
34        public UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
35            ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
36        {
37        }
38    }
39 }

```



```

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

137
138    /// <summary>
139    /// <para>
140    /// Gets the size using the specified node.
141    /// </para>
142    /// <para></para>
143    /// </summary>
144    /// <param name="node">
145    /// <para>The node.</para>
146    /// <para></para>
147    /// </param>
148    /// <returns>
149    /// <para>The ulong</para>
150    /// <para></para>
151    /// </returns>
152    [MethodImpl(MethodImplOptions.AggressiveInlining)]
153    protected override ulong GetSize(ulong node) => Links[node].SizeAsSource;
154
155    /// <summary>
156    /// <para>
157    /// Sets the size using the specified node.
158    /// </para>
159    /// <para></para>
160    /// </summary>
161    /// <param name="node">
162    /// <para>The node.</para>
163    /// <para></para>
164    /// </param>
165    /// <param name="size">
166    /// <para>The size.</para>
167    /// <para></para>
168    /// </param>
169    [MethodImpl(MethodImplOptions.AggressiveInlining)]
170    protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsSource =
    ↪ 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->RootAsSource;
184
185    /// <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 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)
    

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links targets avl balanced tree methods.
10     /// </para>
11     /// </summary>
12     /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
13     public unsafe class UInt64LinksTargetsAvlBalancedTreeMethods :
14         ↪ UInt64LinksAvlBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="UInt64LinksTargetsAvlBalancedTreeMethods"/> instance.
19         /// </para>
20         /// </summary>
21         /// <param name="constants">
22         /// <para>A constants.</para>
23         /// </param>
24         /// <param name="links">
25         /// <para>A links.</para>
26         /// </param>
27         /// <param name="header">
28         /// <para>A header.</para>
29         /// </param>
30         public UInt64LinksTargetsAvlBalancedTreeMethods(LinksConstants<ulong> constants,
31             ↪ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
32         { }
33
34         /// <summary>
35         /// <para>
36         /// Gets the left reference using the specified node.
37         /// </para>
38         /// </summary>
39         /// <param name="node">
40         /// <para>The node.</para>
41         /// </param>
42         /// <returns>
43         /// <para>The ref ulong</para>
44         /// </returns>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

```

126     /// </summary>
127     /// <param name="node">
128     /// <para>The node.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="right">
132     /// <para>The right.</para>
133     /// <para></para>
134     /// </param>
135     [MethodImpl(MethodImplOptions.AggressiveInlining)]
136     protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
        ↳ right;

137     /// <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) => GetSizeValue(Links[node].SizeAsTarget);

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) => SetSizeValue(ref
        ↳ Links[node].SizeAsTarget, size);

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

188     /// <summary>
189     /// <para>
190     /// Sets the left is child using the specified node.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     /// <param name="node">
195     /// <para>The node.</para>
196     /// <para></para>
197     /// </param>
198     /// <param name="value">
199     /// <para>The value.</para>
200     /// <para></para>

```

```

201 /// <para></para>
202 /// </param>
203 [MethodImpl(MethodImplOptions.AggressiveInlining)]
204 protected override void SetLeftIsChild(ulong node, bool value) =>
205     ↪ SetLeftIsChildValue(ref Links[node].SizeAsTarget, value);
206
207 /// <summary>
208 /// <para>
209 /// Determines whether this instance get right is child.
210 /// </para>
211 /// </summary>
212 /// <param name="node">
213 /// <para>The node.</para>
214 /// </para>
215 /// </param>
216 /// <returns>
217 /// <para>The bool</para>
218 /// </returns>
219 [MethodImpl(MethodImplOptions.AggressiveInlining)]
220 protected override bool GetRightIsChild(ulong node) =>
221     ↪ GetRightIsChildValue(Links[node].SizeAsTarget);
222
223 /// <summary>
224 /// <para>
225 /// Sets the right is child using the specified node.
226 /// </para>
227 /// </summary>
228 /// <param name="node">
229 /// <para>The node.</para>
230 /// </para>
231 /// </param>
232 /// <param name="value">
233 /// <para>The value.</para>
234 /// </param>
235 [MethodImpl(MethodImplOptions.AggressiveInlining)]
236 protected override void SetRightIsChild(ulong node, bool value) =>
237     ↪ SetRightIsChildValue(ref Links[node].SizeAsTarget, value);
238
239 /// <summary>
240 /// <para>
241 /// Gets the balance using the specified node.
242 /// </para>
243 /// </summary>
244 /// <param name="node">
245 /// <para>The node.</para>
246 /// </para>
247 /// </param>
248 /// <returns>
249 /// <para>The sbyte</para>
250 /// </returns>
251 [MethodImpl(MethodImplOptions.AggressiveInlining)]
252 protected override sbyte GetBalance(ulong node) =>
253     ↪ GetBalanceValue(Links[node].SizeAsTarget);
254
255 /// <summary>
256 /// <para>
257 /// Sets the balance using the specified node.
258 /// </para>
259 /// </summary>
260 /// <param name="node">
261 /// <para>The node.</para>
262 /// </para>
263 /// </param>
264 /// <param name="value">
265 /// <para>The value.</para>
266 /// </param>
267 [MethodImpl(MethodImplOptions.AggressiveInlining)]
268 protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
269     ↪ Links[node].SizeAsTarget, value);

```

```

273     /// <summary>
274     /// <para>
275     /// Gets the tree root.
276     /// </para>
277     /// <para></para>
278     /// </summary>
279     /// <returns>
280     /// <para>The ulong</para>
281     /// <para></para>
282     /// </returns>
283     [MethodImpl(MethodImplOptions.AggressiveInlining)]
284     protected override ulong GetTreeRoot() => Header->RootAsTarget;
285
286     /// <summary>
287     /// <para>
288     /// Gets the base part value using the specified link.
289     /// </para>
290     /// <para></para>
291     /// </summary>
292     /// <param name="link">
293     /// <para>The link.</para>
294     /// <para></para>
295     /// </param>
296     /// <returns>
297     /// <para>The ulong</para>
298     /// <para></para>
299     /// </returns>
300     [MethodImpl(MethodImplOptions.AggressiveInlining)]
301     protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
302
303     /// <summary>
304     /// <para>
305     /// Determines whether this instance first is to the left of second.
306     /// </para>
307     /// <para></para>
308     /// </summary>
309     /// <param name="firstSource">
310     /// <para>The first source.</para>
311     /// <para></para>
312     /// </param>
313     /// <param name="firstTarget">
314     /// <para>The first target.</para>
315     /// <para></para>
316     /// </param>
317     /// <param name="secondSource">
318     /// <para>The second source.</para>
319     /// <para></para>
320     /// </param>
321     /// <param name="secondTarget">
322     /// <para>The second target.</para>
323     /// <para></para>
324     /// </param>
325     /// <returns>
326     /// <para>The bool</para>
327     /// <para></para>
328     /// </returns>
329     [MethodImpl(MethodImplOptions.AggressiveInlining)]
330     protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
331     ↪     ulong secondSource, ulong secondTarget)
332     ↪     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
333     ↪     secondSource);
334
335     /// <summary>
336     /// <para>
337     /// Determines whether this instance first is to the right of second.
338     /// </para>
339     /// <para></para>
340     /// </summary>
341     /// <param name="firstSource">
342     /// <para>The first source.</para>
343     /// <para></para>
344     /// </param>
345     /// <param name="firstTarget">
346     /// <para>The first target.</para>
347     /// <para></para>
348     /// </param>
349     /// <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(ulong firstSource, ulong firstTarget,
362         ↪ ulong secondSource, ulong secondTarget)
363         => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
364             ↪ secondSource);
365
366     /// <summary>
367     /// <para>
368     /// Clears the node using the specified node.
369     /// </para>
370     /// </summary>
371     /// <param name="node">
372     /// <para>The node.</para>
373     /// <para></para>
374     /// </param>
375     [MethodImpl(MethodImplOptions.AggressiveInlining)]
376     protected override void ClearNode(ulong node)
377     {
378         ref var link = ref Links[node];
379         link.LeftAsTarget = OUL;
380         link.RightAsTarget = OUL;
381         link.SizeAsTarget = OUL;
382     }
383 }

```

1.108 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethodsBase

```

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="UInt64LinksTargetsRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods :
15         ↪ UInt64LinksTargetsRecursionlessSizeBalancedTreeMethodsBase
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     }

```

```

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

```

```

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

```

```

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

```

```

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 }

```

1.109 ./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>
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 UInt64LinksTargetsSizeBalancedTreeMethods(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;

```

```

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

```

```

128     /// <para>The node.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="right">
132     /// <para>The right.</para>
133     /// <para></para>
134     /// </param>
135     [MethodImpl(MethodImplOptions.AggressiveInlining)]
136     protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
        ↪ right;

137
138     /// <summary>
139     /// <para>
140     /// Gets the size using the specified node.
141     /// </para>
142     /// <para></para>
143     /// </summary>
144     /// <param name="node">
145     /// <para>The node.</para>
146     /// <para></para>
147     /// </param>
148     /// <returns>
149     /// <para>The ulong</para>
150     /// <para></para>
151     /// </returns>
152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
153     protected override ulong GetSize(ulong node) => Links[node].SizeAsTarget;
154
155     /// <summary>
156     /// <para>
157     /// Sets the size using the specified node.
158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="node">
162     /// <para>The node.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// <para></para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsTarget =
        ↪ size;

171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// <para></para>
177     /// </summary>
178     /// <returns>
179     /// <para>The ulong</para>
180     /// <para></para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     protected override ulong GetTreeRoot() => Header->RootAsTarget;
184
185     /// <summary>
186     /// <para>
187     /// Gets the base part value using the specified link.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="link">
192     /// <para>The link.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The ulong</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
201
202     /// <summary>
203     /// <para>

```

```

/// 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)
    => firstTarget < secondTarget || (firstTarget == secondTarget && 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>
/// <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)
    => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
    ↪ secondSource);

/// <summary>
/// <para>
/// Clears the node using the specified node.
/// </para>
/// <para></para>
/// </summary>
/// <param name="node">
/// <para>The node.</para>
/// <para></para>
/// </param>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override void ClearNode(ulong node)
{
    ref var link = ref Links[node];
    link.LeftAsTarget = OUL;
    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
13     ///     organizing the storage of links with addresses represented as <see cref="ulong">
14     ///     </para>
15     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
16     ///     размером, для организации хранения связей с адресами представленными в виде <see
17     ///     cref="ulong"> </para>
18     /// </summary>
19     public unsafe class UInt64UnitedMemoryLinks : UnitedMemoryLinksBase<ulong>
20     {
21         private readonly Func<ILinksTreeMethods<ulong>> _createSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<ulong>> _createTargetTreeMethods;
23         private LinksHeader<ulong>* _header;
24         private RawLink<ulong>* _links;
25
26         /// <summary>
27         /// <para>
28         ///     Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="address">
33         /// <para>A address.</para>
34         /// <para></para>
35         /// </param>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public UInt64UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
38
39         /// <summary>
40         /// <para>Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
41         ///     минимальным шагом расширения базы данных.
42         /// </para>
43         /// </summary>
44         /// <param name="address">Полный путь к файлу базы данных.</param>
45         /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
46         ///     байтах.</param>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public UInt64UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
49         FileMappedResizableDirectMemory(address, memoryReservationStep),
50         memoryReservationStep) { }
51
52         /// <summary>
53         /// <para>
54         ///     Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <param name="memory">
59         /// <para>A memory.</para>
60         /// <para></para>
61         /// </param>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public UInt64UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
64         DefaultLinksSizeStep) { }
65
66         /// <summary>
67         /// <para>
68         ///     Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
69         /// </para>
70         /// <para></para>
71         /// </summary>
72         /// <param name="memory">
73         /// <para>A memory.</para>
74         /// <para></para>
75         /// </param>

```

```

65     /// </param>
66     /// <param name="memoryReservationStep">
67     /// <para>A memory reservation step.</para>
68     /// <para></para>
69     /// </param>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep) : this(memory, memoryReservationStep,
        ↳ Default<LinksConstants<ulong>>.Instance, IndexTreeType.Default) { }

72
73     /// <summary>
74     /// <para>
75     /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> 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 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>

```

```

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

```

```

285     /// <summary>
286     /// <para>
287     /// Gets the zero.
288     /// </para>
289     /// <para></para>
290     /// </summary>
291     /// <returns>
292     /// <para>The ulong</para>
293     /// <para></para>
294     /// </returns>
295     [MethodImpl(MethodImplOptions.AggressiveInlining)]
296     protected override ulong GetZero() => 0UL;
297
298     /// <summary>
299     /// <para>
300     /// Gets the one.
301     /// </para>
302     /// <para></para>
303     /// </summary>
304     /// <returns>
305     /// <para>The ulong</para>
306     /// <para></para>
307     /// </returns>
308     [MethodImpl(MethodImplOptions.AggressiveInlining)]
309     protected override ulong GetOne() => 1UL;
310
311     /// <summary>
312     /// <para>
313     /// Converts the to int 64 using the specified value.
314     /// </para>
315     /// <para></para>
316     /// </summary>
317     /// <param name="value">
318     /// <para>The value.</para>
319     /// <para></para>
320     /// </param>
321     /// <returns>
322     /// <para>The long</para>
323     /// <para></para>
324     /// </returns>
325     [MethodImpl(MethodImplOptions.AggressiveInlining)]
326     protected override long ConvertToInt64(ulong value) => (long)value;
327
328     /// <summary>
329     /// <para>
330     /// Converts the to address using the specified value.
331     /// </para>
332     /// <para></para>
333     /// </summary>
334     /// <param name="value">
335     /// <para>The value.</para>
336     /// <para></para>
337     /// </param>
338     /// <returns>
339     /// <para>The ulong</para>
340     /// <para></para>
341     /// </returns>
342     [MethodImpl(MethodImplOptions.AggressiveInlining)]
343     protected override ulong ConvertToAddress(long value) => (ulong)value;
344
345     /// <summary>
346     /// <para>
347     /// Adds the first.
348     /// </para>
349     /// <para></para>
350     /// </summary>
351     /// <param name="first">
352     /// <para>The first.</para>
353     /// <para></para>
354     /// </param>
355     /// <param name="second">
356     /// <para>The second.</para>
357     /// <para></para>
358     /// </param>
359     /// <returns>
360     /// <para>The ulong</para>
361     /// <para></para>
362     /// </returns>

```

```

363 [MethodImpl(MethodImplOptions.AggressiveInlining)]
364 protected override ulong Add(ulong first, ulong second) => first + second;
365
366 /// <summary>
367 /// <para>
368 /// Subtracts the first.
369 /// </para>
370 /// <para></para>
371 /// </summary>
372 /// <param name="first">
373 /// <para>The first.</para>
374 /// <para></para>
375 /// </param>
376 /// <param name="second">
377 /// <para>The second.</para>
378 /// <para></para>
379 /// </param>
380 /// <returns>
381 /// <para>The ulong</para>
382 /// <para></para>
383 /// </returns>
384 [MethodImpl(MethodImplOptions.AggressiveInlining)]
385 protected override ulong Subtract(ulong first, ulong second) => first - second;
386
387 /// <summary>
388 /// <para>
389 /// Increments the link.
390 /// </para>
391 /// <para></para>
392 /// </summary>
393 /// <param name="link">
394 /// <para>The link.</para>
395 /// <para></para>
396 /// </param>
397 /// <returns>
398 /// <para>The ulong</para>
399 /// <para></para>
400 /// </returns>
401 [MethodImpl(MethodImplOptions.AggressiveInlining)]
402 protected override ulong Increment(ulong link) => ++link;
403
404 /// <summary>
405 /// <para>
406 /// Decrements the link.
407 /// </para>
408 /// <para></para>
409 /// </summary>
410 /// <param name="link">
411 /// <para>The link.</para>
412 /// <para></para>
413 /// </param>
414 /// <returns>
415 /// <para>The ulong</para>
416 /// <para></para>
417 /// </returns>
418 [MethodImpl(MethodImplOptions.AggressiveInlining)]
419 protected override ulong Decrement(ulong link) => --link;
420 }
421 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 unused links list methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UnusedLinksListMethods{ulong}">
15    public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<ulong>
16    {
17        private readonly RawLink<ulong>* _links;
18        private readonly LinksHeader<ulong>* _header;

```

```

19
20     /// <summary>
21     /// <para>
22     /// Initializes a new <see cref="UInt64UnusedLinksListMethods"/> instance.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public UInt64UnusedLinksListMethods(RawLink<ulong>* links, LinksHeader<ulong>* header)
36         : base((byte*)links, (byte*)header)
37     {
38         _links = links;
39         _header = header;
40     }
41
42     /// <summary>
43     /// <para>
44     /// Gets the link reference using the specified link.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="link">
49     /// <para>The link.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>A ref raw link of ulong</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref _links[link];
58
59     /// <summary>
60     /// <para>
61     /// Gets the header reference.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <returns>
66     /// <para>A ref links header of ulong</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
71 }
72 }

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.PropertyOperators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the properties operator.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLinkAddress}"/>
16     /// <seealso cref="IProperties{TLinkAddress, TLinkAddress, TLinkAddress}"/>
17     public class PropertiesOperator<TLinkAddress> : LinksOperatorBase<TLinkAddress>,
18         ↪ IProperties<TLinkAddress, TLinkAddress, TLinkAddress> where TLinkAddress : struct
19     {
20         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
21         ↪ EqualityComparer<TLinkAddress>.Default;
22
23         /// <summary>

```

```

22     /// <para>
23     /// Initializes a new <see cref="PropertiesOperator"/> 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 PropertiesOperator(ILinks<TLinkAddress> links) : base(links) { }
33
34     /// <summary>
35     /// <para>
36     /// Gets the value using the specified object.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     /// <param name="@object">
41     /// <para>The object.</para>
42     /// <para></para>
43     /// </param>
44     /// <param name="property">
45     /// <para>The property.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public TLinkAddress GetValue(TLinkAddress @object, TLinkAddress 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<TLinkAddress>(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(TLinkAddress @object, TLinkAddress property, TLinkAddress 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{TLinkAddress}"/>
16     /// <seealso cref="IProperty{TLinkAddress, TLinkAddress}"/>
17     public class PropertyOperator<TLinkAddress> : LinksOperatorBase<TLinkAddress>,
18     ↪ IProperty<TLinkAddress, TLinkAddress> where TLinkAddress : struct
19     {
20         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
21         ↪ EqualityComparer<TLinkAddress>.Default;
22         private readonly TLinkAddress _propertyMarker;
23         private readonly TLinkAddress _propertyValueMarker;
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="PropertyOperator"/> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="links">
32         /// <para>A links.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="propertyMarker">
36         /// <para>A property marker.</para>
37         /// <para></para>
38         /// </param>
39         /// <param name="propertyValueMarker">
40         /// <para>A property value marker.</para>
41         /// <para></para>
42         /// </param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public PropertyOperator(ILinks<TLinkAddress> links, TLinkAddress propertyMarker,
45         ↪ TLinkAddress propertyValueMarker) : base(links)
46         {
47             _propertyMarker = propertyMarker;
48             _propertyValueMarker = propertyValueMarker;
49         }
50
51         /// <summary>
52         /// <para>
53         /// Gets the link.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         /// <param name="link">
58         /// <para>The link.</para>
59         /// <para></para>
60         /// </param>
61         /// <returns>
62         /// <para>The link</para>
63         /// <para></para>
64         /// </returns>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public TLinkAddress Get(TLinkAddress link)
67         {
68             var property = _links.SearchOrDefault(link, _propertyMarker);
69             return GetValue(GetContainer(property));
70         }
71
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         private TLinkAddress GetContainer(TLinkAddress property)
74         {
75             var valueContainer = default(TLinkAddress);
76             if (_equalityComparer.Equals(property, default))
77             {
78                 return valueContainer;
79             }
80         }
81     }
82 }

```

```

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<TLinkAddress>(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 TLinkAddress GetValue(TLinkAddress container) =>
97     ↪ _equalityComparer.Equals(container, default) ? default : _links.GetTarget(container);
98
99 /// <summary>
100 /// <para>
101 /// Sets the link.
102 /// </para>
103 /// <para></para>
104 /// </summary>
105 /// <param name="link">
106 /// <para>The link.</para>
107 /// <para></para>
108 /// </param>
109 /// <param name="value">
110 /// <para>The value.</para>
111 /// <para></para>
112 /// </param>
113 [MethodImpl(MethodImplOptions.AggressiveInlining)]
114 public void Set(TLinkAddress link, TLinkAddress value)
115 {
116     var links = _links;
117     var property = links.GetOrCreate(link, _propertyMarker);
118     var container = GetContainer(property);
119     if (_equalityComparer.Equals(container, default))
120     {
121         links.GetOrCreate(property, value);
122     }
123     else
124     {
125         links.Update(container, property, value);
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{TLinkAddress}" />
17     /// <seealso cref="IStack{TLinkAddress}" />
18     public class Stack<TLinkAddress> : LinksOperatorBase<TLinkAddress>, IStack<TLinkAddress>
19     {
20         ↪ where TLinkAddress : struct
21         {
22             private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
23                 ↪ EqualityComparer<TLinkAddress>.Default;
24             private readonly TLinkAddress _stack;
25         }
26     }
27 }

```

```

22
23     /// <summary>
24     /// <para>
25     /// Gets the is empty value.
26     /// </para>
27     /// <para></para>
28     /// </summary>
29     public bool IsEmpty
30     {
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         get => _equalityComparer.Equals(Peek(), _stack);
33     }
34
35     /// <summary>
36     /// <para>
37     /// Initializes a new <see cref="Stack"/> instance.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     /// <param name="links">
42     /// <para>A links.</para>
43     /// <para></para>
44     /// </param>
45     /// <param name="stack">
46     /// <para>A stack.</para>
47     /// <para></para>
48     /// </param>
49     [MethodImpl(MethodImplOptions.AggressiveInlining)]
50     public Stack(ILinks<TLinkAddress> links, TLinkAddress stack) : base(links) => _stack =
51     => stack;
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     private TLinkAddress GetStackMarker() => _links.GetSource(_stack);
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     private TLinkAddress GetTop() => _links.GetTarget(_stack);
56
57     /// <summary>
58     /// <para>
59     /// Peeks this instance.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <returns>
64     /// <para>The link</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public TLinkAddress Peek() => _links.GetTarget(GetTop());
69
70     /// <summary>
71     /// <para>
72     /// Pops this instance.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <returns>
77     /// <para>The element.</para>
78     /// <para></para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public TLinkAddress Pop()
82     {
83         var element = Peek();
84         if (!_equalityComparer.Equals(element, _stack))
85         {
86             var top = GetTop();
87             var previousTop = _links.GetSource(top);
88             _links.Update(_stack, GetStackMarker(), previousTop);
89             _links.Delete(top);
90         }
91         return element;
92     }
93
94     /// <summary>
95     /// <para>
96     /// Pushes the element.
97     /// </para>
98     /// <para></para>
99     /// </summary>

```

```

99     /// <param name="element">
100     /// <para>The element.</para>
101     /// <para></para>
102     /// </param>
103     [MethodImpl(MethodImplOptions.AggressiveInlining)]
104     public void Push(TLinkAddress element) => _links.Update(_stack, GetStackMarker(),
        ↪ _links.GetOrCreate(GetTop(), element));
105 }
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="TLinkAddress">
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 TLinkAddress CreateStack<TLinkAddress>(this ILinks<TLinkAddress> links,
            ↪ TLinkAddress stackMarker) where TLinkAddress : struct
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> where
        ↪ TLinkAddress : struct
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     {

```

```

97         SyncRoot = synchronization;
98         Sync = this;
99         Unsync = links;
100        Constants = links.Constants;
101    }
102
103    /// <summary>
104    /// <para>
105    /// Counts the restriction.
106    /// </para>
107    /// <para></para>
108    /// </summary>
109    /// <param name="restriction">
110    /// <para>The restriction.</para>
111    /// <para></para>
112    /// </param>
113    /// <returns>
114    /// <para>The link address</para>
115    /// <para></para>
116    /// </returns>
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    public TLinkAddress Count(IList<TLinkAddress>? restriction) =>
119        ↳ SyncRoot.ExecuteReadOperation(restriction, Unsync.Count);
120
121    /// <summary>
122    /// <para>
123    /// Eaches the handler.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="handler">
128    /// <para>The handler.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="restriction">
132    /// <para>The substitution.</para>
133    /// <para></para>
134    /// </param>
135    /// <returns>
136    /// <para>The link address</para>
137    /// <para></para>
138    /// </returns>
139    [MethodImpl(MethodImplOptions.AggressiveInlining)]
140    public TLinkAddress Each(IList<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
141        ↳ handler) => SyncRoot.ExecuteReadOperation(restriction, handler, Unsync.Each);
142
143    /// <summary>
144    /// <para>
145    /// Creates the substitution.
146    /// </para>
147    /// <para></para>
148    /// </summary>
149    /// <param name="substitution">
150    /// <para>The substitution.</para>
151    /// <para></para>
152    /// </param>
153    /// <returns>
154    /// <para>The link address</para>
155    /// <para></para>
156    /// </returns>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    public TLinkAddress Create(IList<TLinkAddress>? substitution,
159        ↳ WriteHandler<TLinkAddress>? handler) => SyncRoot.ExecuteWriteOperation(substitution,
160        ↳ handler, Unsync.Create);
161
162    /// <summary>
163    /// <para>
164    /// Updates the substitution.
165    /// </para>
166    /// <para></para>
167    /// </summary>
168    /// <param name="restriction">
169    /// <para>The substitution.</para>
170    /// <para></para>
171    /// </param>
172    /// <param name="substitution">
173    /// <para>The substitution.</para>
174    /// <para></para>
175    /// </param>

```

```

171     /// </param>
172     /// <returns>
173     /// <para>The link address</para>
174     /// <para></para>
175     /// </returns>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     public TLinkAddress Update(IList<TLinkAddress>? restriction, IList<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(IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
        ↳ handler) => SyncRoot.ExecuteWriteOperation(restriction, handler, Unsync.Delete);

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

```

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

```

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

26
27         /// <summary>
28         /// <para>
29         /// Determines whether any link is any.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="links">
34         /// <para>The links.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="sequence">
38         /// <para>The sequence.</para>

```

```

39     /// <para></para>
40     /// </param>
41     /// <returns>
42     /// <para>The bool</para>
43     /// <para></para>
44     /// </returns>
45     [MethodImpl(MethodImplOptions.AggressiveInlining)]
46     public static bool AnyLinkIsAny(this ILinks<ulong> links, params ulong[] sequence)
47     {
48         if (sequence == null)
49         {
50             return false;
51         }
52         var constants = links.Constants;
53         for (var i = 0; i < sequence.Length; i++)
54         {
55             if (sequence[i] == constants.Any)
56             {
57                 return true;
58             }
59         }
60         return false;
61     }
62
63     /// <summary>
64     /// <para>
65     /// Formats the structure using the specified links.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="links">
70     /// <para>The links.</para>
71     /// <para></para>
72     /// </param>
73     /// <param name="linkIndex">
74     /// <para>The link index.</para>
75     /// <para></para>
76     /// </param>
77     /// <param name="isElement">
78     /// <para>The is element.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="renderIndex">
82     /// <para>The render index.</para>
83     /// <para></para>
84     /// </param>
85     /// <param name="renderDebug">
86     /// <para>The render debug.</para>
87     /// <para></para>
88     /// </param>
89     /// <returns>
90     /// <para>The string</para>
91     /// <para></para>
92     /// </returns>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
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>

```



```

114     /// <para></para>
115     /// </param>
116     /// <param name="isElement">
117     /// <para>The is element.</para>
118     /// <para></para>
119     /// </param>
120     /// <param name="appendElement">
121     /// <para>The append element.</para>
122     /// <para></para>
123     /// </param>
124     /// <param name="renderIndex">
125     /// <para>The render index.</para>
126     /// <para></para>
127     /// </param>
128     /// <param name="renderDebug">
129     /// <para>The render debug.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The string</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
    ↪ Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
    ↪ bool renderIndex = false, bool renderDebug = false)
138     {
139         var sb = new StringBuilder();
140         var visited = new HashSet<ulong>();
141         links.AppendStructure(sb, visited, linkIndex, isElement, appendElement, renderIndex,
    ↪ 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 using TLinkAddress = System.UInt64;
16
17 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
18
19 namespace Platform.Data.Doublets
20 {
21     /// <summary>
22     /// <para>
23     /// Represents the int 64 links transactions layer.
24     /// </para>
25     /// <para></para>
26     /// </summary>
27     /// <seealso cref="LinksDisposableDecoratorBase{TLinkAddress}"/>
28     public class UInt64LinksTransactionsLayer : LinksDisposableDecoratorBase<TLinkAddress>
29     {
30         // --V3073 where TLinkAddress : struct
31         {
32             /// <remarks>
33             /// Альтернативные варианты хранения трансформации (элемента транзакции):
34             ///
35             /// private enum TransitionType
36             /// {
37             ///     Creation,
38             ///     UpdateOf,
39             ///     UpdateTo,
40             ///     Deletion
41             /// }
42             ///
43             /// private struct Transition
44             /// {
45             ///     public TLinkAddress TransactionId;
46             ///     public UniqueTimestamp Timestamp;
47             ///     public TransactionItemType Type;
48             ///     public Link Source;
49             ///     public Link Linker;
50             ///     public Link Target;
51             /// }
52             /// Или
53             ///
54             public struct TransitionHeader
55             {
56                 public TLinkAddress TransactionIdCombined;
57                 public TLinkAddress TimestampCombined;
58             }
59             public TLinkAddress TransactionId
60             {
61                 get
62                 {
63                     return (TLinkAddress) mask & TransactionIdCombined;
64                 }
65             }
66             public UniqueTimestamp Timestamp
67             {
68                 get
69                 {
70                     return (UniqueTimestamp) mask & TransactionIdCombined;
71                 }
72             }
73         }
74     }
75 }

```

```

73     ///
74     public TransactionItemType Type
75     {
76         ///     get
77         ///     {
78             ///         // Использовать по одному биту из TransactionId и Timestamp,
79             ///         // для значения в 2 бита, которое представляет тип операции
80             ///         throw new NotImplementedException();
81         }
82     }
83 }
84
85 private struct Transition
86 {
87     public TransitionHeader Header;
88     public Link Source;
89     public Link Linker;
90     public Link Target;
91 }
92
93 </remarks>
94 public struct Transition : IEquatable<Transition>
95 {
96     /// <summary>
97     /// <para>
98     /// The size.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    public static readonly long Size = Structure<Transition>.Size;
103
104    /// <summary>
105    /// <para>
106    /// The transaction id.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    public readonly TLinkAddress TransactionId;
111    /// <summary>
112    /// <para>
113    /// The before.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    public readonly Link<TLinkAddress> Before;
118    /// <summary>
119    /// <para>
120    /// The after.
121    /// </para>
122    /// <para></para>
123    /// </summary>
124    public readonly Link<TLinkAddress> After;
125    /// <summary>
126    /// <para>
127    /// The timestamp.
128    /// </para>
129    /// <para></para>
130    /// </summary>
131    public readonly Timestamp Timestamp;
132
133    /// <summary>
134    /// <para>
135    /// Initializes a new <see cref="Transition"/> instance.
136    /// </para>
137    /// <para></para>
138    /// </summary>
139    /// <param name="uniqueTimestampFactory">
140    /// <para>A unique timestamp factory.</para>
141    /// <para></para>
142    /// </param>
143    /// <param name="transactionId">
144    /// <para>A transaction id.</para>
145    /// <para></para>
146    /// </param>
147    /// <param name="before">
148    /// <para>A before.</para>
149    /// <para></para>
150    /// </param>

```

```

151 /// <param name="after">
152 /// <para>A after.</para>
153 /// <para></para>
154 /// </param>
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 public Transition(UniqueTimestampFactory uniqueTimestampFactory, TLinkAddress
↳ transactionId, Link<TLinkAddress> before, Link<TLinkAddress> after)
157 {
158     TransactionId = transactionId;
159     Before = before;
160     After = after;
161     Timestamp = uniqueTimestampFactory.Create();
162 }
163
164 public Transition(UniqueTimestampFactory uniqueTimestampFactory, TLinkAddress
↳ transactionId, IList<TLinkAddress> before, IList<TLinkAddress> after) :
↳ this(uniqueTimestampFactory, transactionId, new Link<TLinkAddress>(before), new
↳ Link<TLinkAddress>(after)) { }

165
166 /// <summary>
167 /// <para>
168 /// Initializes a new <see cref="Transition"/> instance.
169 /// </para>
170 /// <para></para>
171 /// </summary>
172 /// <param name="uniqueTimestampFactory">
173 /// <para>A unique timestamp factory.</para>
174 /// <para></para>
175 /// </param>
176 /// <param name="transactionId">
177 /// <para>A transaction id.</para>
178 /// <para></para>
179 /// </param>
180 /// <param name="before">
181 /// <para>A before.</para>
182 /// <para></para>
183 /// </param>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 public Transition(UniqueTimestampFactory uniqueTimestampFactory, TLinkAddress
↳ transactionId, Link<TLinkAddress> before) : this(uniqueTimestampFactory,
↳ transactionId, before, default) { }

186
187 /// <summary>
188 /// <para>
189 /// Initializes a new <see cref="Transition"/> instance.
190 /// </para>
191 /// <para></para>
192 /// </summary>
193 /// <param name="uniqueTimestampFactory">
194 /// <para>A unique timestamp factory.</para>
195 /// <para></para>
196 /// </param>
197 /// <param name="transactionId">
198 /// <para>A transaction id.</para>
199 /// <para></para>
200 /// </param>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 public Transition(UniqueTimestampFactory uniqueTimestampFactory, TLinkAddress
↳ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
↳ }

203
204 /// <summary>
205 /// <para>
206 /// Returns the string.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 /// <returns>
211 /// <para>The string</para>
212 /// <para></para>
213 /// </returns>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 public override string ToString() => $"{Timestamp} {TransactionId}: {Before} =>
↳ {After}";

216
217 /// <summary>
218 /// <para>
219 /// Determines whether this instance equals.

```

```

220     /// </para>
221     /// <para></para>
222     /// </summary>
223     /// <param name="obj">
224     /// <para>The obj.</para>
225     /// <para></para>
226     /// </param>
227     /// <returns>
228     /// <para>The bool</para>
229     /// <para></para>
230     /// </returns>
231     [MethodImpl(MethodImplOptions.AggressiveInlining)]
232     public override bool Equals(object obj) => obj is Transition transition ?
        ↳ Equals(transition) : false;

233
234     /// <summary>
235     /// <para>
236     /// Gets the hash code.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <returns>
241     /// <para>The int</para>
242     /// <para></para>
243     /// </returns>
244     [MethodImpl(MethodImplOptions.AggressiveInlining)]
245     public override int GetHashCode() => (TransactionId, Before, After,
        ↳ Timestamp).GetHashCode();

246
247     /// <summary>
248     /// <para>
249     /// Determines whether this instance equals.
250     /// </para>
251     /// <para></para>
252     /// </summary>
253     /// <param name="other">
254     /// <para>The other.</para>
255     /// <para></para>
256     /// </param>
257     /// <returns>
258     /// <para>The bool</para>
259     /// <para></para>
260     /// </returns>
261     [MethodImpl(MethodImplOptions.AggressiveInlining)]
262     public bool Equals(Transition other) => TransactionId == other.TransactionId &&
        ↳ Before == other.Before && After == other.After && Timestamp == other.Timestamp;

263
264     [MethodImpl(MethodImplOptions.AggressiveInlining)]
265     public static bool operator ==(Transition left, Transition right) =>
        ↳ left.Equals(right);

266
267     [MethodImpl(MethodImplOptions.AggressiveInlining)]
268     public static bool operator !=(Transition left, Transition right) => !(left ==
        ↳ right);

269 }

270
271 /// <remarks>
272 /// Другие варианты реализации транзакций (атомарности):
273 /// 1. Разделение хранения значения связи ((Source Target) или (Source Linker
274   ↳ Target)) и индексов.
275 /// 2. Хранение трансформаций/операций в отдельном хранилище Links, но дополнительно
276   ↳ потребуется решить вопрос
277   ↳ со ссылками на внешние идентификаторы, или как-то иначе решить вопрос с
278   ↳ пересечениями идентификаторов.
279 ///
280 /// Где хранить промежуточный список транзакций?
281 ///
282 /// В оперативной памяти:
283 /// Минусы:
284 /// 1. Может усложнить систему, если она будет функционировать самостоятельно,
285   ↳ так как нужно отдельно выделять память под список трансформаций.
286   ↳ 2. Выделенной оперативной памяти может не хватить, в том случае,
287   ↳ если транзакция использует слишком много трансформаций.
288   ↳ -> Можно использовать жёсткий диск для слишком длинных транзакций.
289   ↳ -> Максимальный размер списка трансформаций можно ограничить / задать
290   ↳ константой.
291 /// 3. При подтверждении транзакции (Commit) все трансформации записываются разом
292   ↳ создавая задержку.

```

```

288 ///
289 /// На жёстком диске:
290 /// Минусы:
291 /// 1. Длительный отклик, на запись каждой трансформации.
292 /// 2. Лог транзакций дополнительно наполняется отменёнными транзакциями.
293 /// -> Это может решаться упаковкой/исключением дублирующих операций.
294 /// -> Также это может решаться тем, что короткие транзакции вообще
295 /// не будут записываться в случае отката.
296 /// 3. Перед тем как выполнять отмену операций транзакции нужно дождаться пока все
    -> операции (трансформации)
297 /// будут записаны в лог.
298 ///
299 </remarks>
300 public class Transaction : DisposableBase
301 {
302     private readonly Queue<Transition> _transitions;
303     private readonly UInt64LinksTransactionsLayer _layer;
304     /// <summary>
305     /// <para>
306     /// Gets or sets the is committed value.
307     /// </para>
308     /// <para></para>
309     /// </summary>
310     public bool IsCommitted { get; private set; }
311     /// <summary>
312     /// <para>
313     /// Gets or sets the is reverted value.
314     /// </para>
315     /// <para></para>
316     /// </summary>
317     public bool IsReverted { get; private set; }
318
319     /// <summary>
320     /// <para>
321     /// Initializes a new <see cref="Transaction"/> instance.
322     /// </para>
323     /// <para></para>
324     /// </summary>
325     /// <param name="layer">
326     /// <para>A layer.</para>
327     /// <para></para>
328     /// </param>
329     /// <exception cref="NotSupportedException">
330     /// <para>Nested transactions not supported.</para>
331     /// <para></para>
332     /// </exception>
333     [MethodImpl(MethodImplOptions.AggressiveInlining)]
334     public Transaction(UInt64LinksTransactionsLayer layer)
335     {
336         _layer = layer;
337         if (_layer._currentTransactionId != 0)
338         {
339             throw new NotSupportedException("Nested transactions not supported.");
340         }
341         IsCommitted = false;
342         IsReverted = false;
343         _transitions = new Queue<Transition>();
344         SetCurrentTransaction(layer, this);
345     }
346
347     /// <summary>
348     /// <para>
349     /// Commits this instance.
350     /// </para>
351     /// <para></para>
352     /// </summary>
353     [MethodImpl(MethodImplOptions.AggressiveInlining)]
354     public void Commit()
355     {
356         EnsureTransactionAllowsWriteOperations(this);
357         while (_transitions.Count > 0)
358         {
359             var transition = _transitions.Dequeue();
360             _layer._transitions.Enqueue(transition);
361         }
362         _layer._lastCommittedTransactionId = _layer._currentTransactionId;
363         IsCommitted = true;
364     }

```

```

365 [MethodImpl(MethodImplOptions.AggressiveInlining)]
366 private void Revert()
367 {
368     EnsureTransactionAllowsWriteOperations(this);
369     var transitionsToRevert = new Transition[_transitions.Count];
370     _transitions.CopyTo(transitionsToRevert, 0);
371     for (var i = transitionsToRevert.Length - 1; i >= 0; i--)
372     {
373         _layer.RevertTransition(transitionsToRevert[i]);
374     }
375     IsReverted = true;
376 }
377
378 /// <summary>
379 /// <para>
380 /// Sets the current transaction using the specified layer.
381 /// </para>
382 /// </summary>
383 /// <param name="layer">
384 /// <para>The layer.</para>
385 /// </param>
386 /// <param name="transaction">
387 /// <para>The transaction.</para>
388 /// </param>
389 [MethodImpl(MethodImplOptions.AggressiveInlining)]
390 public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
391     ↪ Transaction transaction)
392 {
393     layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
394     layer._currentTransactionTransitions = transaction._transitions;
395     layer._currentTransaction = transaction;
396 }
397
398 /// <summary>
399 /// <para>
400 /// Ensures the transaction allows write operations using the specified transaction.
401 /// </para>
402 /// </summary>
403 /// <param name="transaction">
404 /// <para>The transaction.</para>
405 /// </param>
406 /// <exception cref="InvalidOperationException">
407 /// <para>Transation is committed.</para>
408 /// </exception>
409 /// <exception cref="InvalidOperationException">
410 /// <para>Transation is reverted.</para>
411 /// </exception>
412 [MethodImpl(MethodImplOptions.AggressiveInlining)]
413 public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
414 {
415     if (transaction.IsReverted)
416     {
417         throw new InvalidOperationException("Transation is reverted.");
418     }
419     if (transaction.IsCommitted)
420     {
421         throw new InvalidOperationException("Transation is committed.");
422     }
423 }
424
425 /// <summary>
426 /// <para>
427 /// Disposes the manual.
428 /// </para>
429 /// </summary>
430 /// <param name="manual">
431 /// <para>The manual.</para>
432 /// </param>
433 /// <param name="wasDisposed">

```



```

442     /// <para>The was disposed.</para>
443     /// <para></para>
444     /// </param>
445     [MethodImpl(MethodImplOptions.AggressiveInlining)]
446     protected override void Dispose(bool manual, bool wasDisposed)
447     {
448         if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
449         {
450             if (!IsCommitted && !IsReverted)
451             {
452                 Revert();
453             }
454             _layer.ResetCurrentTransation();
455         }
456     }
457 }
458
459     /// <summary>
460     /// <para>
461     /// The from seconds.
462     /// </para>
463     /// <para></para>
464     /// </summary>
465     public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
466     private readonly string _logAddress;
467     private readonly FileStream _log;
468     private readonly Queue<Transition> _transitions;
469     private readonly UniqueTimestampFactory _uniqueTimestampFactory;
470     private Task _transitionsPusher;
471     private Transition _lastCommittedTransition;
472     private TLinkAddress _currentTransactionId;
473     private Queue<Transition> _currentTransactionTransitions;
474     private Transaction _currentTransaction;
475     private TLinkAddress _lastCommittedTransactionId;
476
477     /// <summary>
478     /// <para>
479     /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.
480     /// </para>
481     /// <para></para>
482     /// </summary>
483     /// <param name="links">
484     /// <para>A links.</para>
485     /// <para></para>
486     /// </param>
487     /// <param name="logAddress">
488     /// <para>A log address.</para>
489     /// <para></para>
490     /// </param>
491     /// <exception cref="ArgumentNullException">
492     /// <para></para>
493     /// <para></para>
494     /// </exception>
495     /// <exception cref="NotSupportedException">
496     /// <para>Database is damaged, autorecovery is not supported yet.</para>
497     /// <para></para>
498     /// </exception>
499     [MethodImpl(MethodImplOptions.AggressiveInlining)]
500     public UInt64LinksTransactionsLayer(ILinks<TLinkAddress> links, string logAddress)
501         : base(links)
502     {
503         if (string.IsNullOrEmpty(logAddress))
504         {
505             throw new ArgumentNullException(nameof(logAddress));
506         }
507         // В первой строке файла хранится последняя законченную транзакцию.
508         // При запуске это используется для проверки удачного закрытия файла лога.
509         // In the first line of the file the last committed transaction is stored.
510         // On startup, this is used to check that the log file is successfully closed.
511         var lastCommittedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
512         var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
513         if (!lastCommittedTransition.Equals(lastWrittenTransition))
514         {
515             Dispose();
516             throw new NotSupportedException("Database is damaged, autorecovery is not
517                 ↪ supported yet.");
518         }
519         if (lastCommittedTransition == default)

```

```

519     {
520         FileHelpers.WriteFirst(logAddress, lastCommittedTransition);
521     }
522     _lastCommittedTransition = lastCommittedTransition;
523     // TODO: Think about a better way to calculate or store this value
524     var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
525     _lastCommittedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
526         ↪ x.TransactionId) : 0;
527     _uniqueTimestampFactory = new UniqueTimestampFactory();
528     _logAddress = logAddress;
529     _log = FileHelpers.Append(logAddress);
530     _transitions = new Queue<Transition>();
531     _transitionsPusher = new Task(TransitionsPusher);
532     _transitionsPusher.Start();
533 }
534
535 /// <summary>
536 /// <para>
537 /// Gets the link value using the specified link.
538 /// </para>
539 /// </summary>
540 /// <param name="link">
541 /// <para>The link.</para>
542 /// </param>
543 /// <returns>
544 /// <para>A list of TLinkAddress</para>
545 /// </returns>
546 [MethodImpl(MethodImplOptions.AggressiveInlining)]
547 public IList<TLinkAddress> GetLinkValue(TLinkAddress link) => _links.GetLink(link);
548
549 /// <summary>
550 /// <para>
551 /// Creates the substitution.
552 /// </para>
553 /// </summary>
554 /// <param name="substitution">
555 /// <para>The substitution.</para>
556 /// </param>
557 /// <returns>
558 /// <para>The created link index.</para>
559 /// </returns>
560 [MethodImpl(MethodImplOptions.AggressiveInlining)]
561 public override TLinkAddress Create(IList<TLinkAddress>? substitution,
562     ↪ WriteHandler<TLinkAddress>? handler)
563 {
564     return _links.Create(new Link<TLinkAddress>(), (before, after) =>
565     {
566         CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
567             ↪ new Link<TLinkAddress>(before), new Link<TLinkAddress>(after)));
568         return handler?.Invoke(before, after) ?? Links.Constants.Continue;
569     });
570 }
571
572 /// <summary>
573 /// <para>
574 /// Updates the substitution.
575 /// </para>
576 /// </summary>
577 /// <param name="restriction">
578 /// <para>The substitution.</para>
579 /// </param>
580 /// <param name="substitution">
581 /// <para>The substitution.</para>
582 /// </param>
583 /// <returns>
584 /// <para>The link index.</para>
585 /// </returns>
586 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

594 public override TLinkAddress Update(IList<TLinkAddress>? restriction,
    ↳ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
595 {
596     return _links.Update(restriction, substitution, (before, after) =>
597     {
598         CommitTransition(new Transition(_uniqueTimestampFactory,
    ↳ _currentTransactionId, new Link<TLinkAddress>(before), new
    ↳ Link<TLinkAddress>(after)));
        return handler?.Invoke(before, after) ?? Constants.Continue;
    });
    });
601 }
602
603
604 /// <summary>
605 /// <para>
606 /// Deletes the substitution.
607 /// </para>
608 /// <para></para>
609 /// </summary>
610 /// <param name="restriction">
611 /// <para>The substitution.</para>
612 /// </param>
613 [MethodImpl(MethodImplOptions.AggressiveInlining)]
614 public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
    ↳ WriteHandler<TLinkAddress>? handler)
615 {
616     var link = restriction[_constants.IndexPart];
617     return _links.Delete(restriction, (before, after) =>
618     {
619         CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
    ↳ before, after));
        return handler?.Invoke(before, after) ?? Constants.Continue;
    });
622 }
623
624 [MethodImpl(MethodImplOptions.AggressiveInlining)]
625 private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
    ↳ _transitions;
626 [MethodImpl(MethodImplOptions.AggressiveInlining)]
627 private void CommitTransition(Transition transition)
628 {
629     if (_currentTransaction != null)
630     {
631         Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
632     }
633     var transitions = GetCurrentTransitions();
634     transitions.Enqueue(transition);
635 }
636 [MethodImpl(MethodImplOptions.AggressiveInlining)]
637 private void RevertTransition(Transition transition)
638 {
639     if (transition.After.IsNull()) // Revert Deletion with Creation
640     {
641         _links.Create();
642     }
643     else if (transition.Before.IsNull()) // Revert Creation with Deletion
644     {
645         _links.Delete(transition.After.Index);
646     }
647     else // Revert Update
648     {
649         _links.Update(new[] { transition.After.Index, transition.Before.Source,
    ↳ transition.Before.Target });
650     }
651 }
652 [MethodImpl(MethodImplOptions.AggressiveInlining)]
653 private void ResetCurrentTransation()
654 {
655     _currentTransactionId = 0;
656     _currentTransactionTransitions = null;
657     _currentTransaction = null;
658 }
659 [MethodImpl(MethodImplOptions.AggressiveInlining)]
660 private void PushTransitions()
661 {
662     if (_log == null || _transitions == null)
663     {
664         return;

```

```

665     }
666     for (var i = 0; i < _transitions.Count; i++)
667     {
668         var transition = _transitions.Dequeue();
669
670         _log.Write(transition);
671         _lastCommittedTransition = transition;
672     }
673 }
674 [MethodImpl(MethodImplOptions.AggressiveInlining)]
675 private void TransitionsPusher()
676 {
677     while (!Disposable.IsDisposed && _transitionsPusher != null)
678     {
679         Thread.Sleep(DefaultPushDelay);
680         PushTransitions();
681     }
682 }
683
684 /// <summary>
685 /// <para>
686 /// Begins the transaction.
687 /// </para>
688 /// <para></para>
689 /// </summary>
690 /// <returns>
691 /// <para>The transaction</para>
692 /// <para></para>
693 /// </returns>
694 [MethodImpl(MethodImplOptions.AggressiveInlining)]
695 public Transaction BeginTransaction() => new Transaction(this);
696 [MethodImpl(MethodImplOptions.AggressiveInlining)]
697 private void DisposeTransitions()
698 {
699     try
700     {
701         var pusher = _transitionsPusher;
702         if (pusher != null)
703         {
704             _transitionsPusher = null;
705             pusher.Wait();
706         }
707         if (_transitions != null)
708         {
709             PushTransitions();
710         }
711         _log.DisposeIfPossible();
712         FileHelpers.WriteFirst(_logAddress, _lastCommittedTransition);
713     }
714     catch (Exception ex)
715     {
716         ex.Ignore();
717     }
718 }
719
720 #region DisposalBase
721
722 /// <summary>
723 /// <para>
724 /// Disposes the manual.
725 /// </para>
726 /// <para></para>
727 /// </summary>
728 /// <param name="manual">
729 /// <para>The manual.</para>
730 /// <para></para>
731 /// </param>
732 /// <param name="wasDisposed">
733 /// <para>The was disposed.</para>
734 /// <para></para>
735 /// </param>
736 [MethodImpl(MethodImplOptions.AggressiveInlining)]
737 protected override void Dispose(bool manual, bool wasDisposed)
738 {
739     if (!wasDisposed)
740     {
741         DisposeTransitions();
742     }

```

```

743         base.Dispose(manual, wasDisposed);
744     }
745
746     #endregion
747 }
748 }

```

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

```

1  using System;
2  using System.IO;
3  using Platform.Data.Doublets.Decorators;
4  using Xunit;
5
6  using Platform.Memory;
7
8  using Platform.Data.Doublets.Memory.United.Generic;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     public static class GenericLinksTests
13     {
14         [Fact]
15         public static void CRUDTest()
16         {
17             Using<byte>(links => links.TestCRUDOperations());
18             Using<ushort>(links => links.TestCRUDOperations());
19             Using<uint>(links => links.TestCRUDOperations());
20             Using<ulong>(links => links.TestCRUDOperations());
21         }
22
23         [Fact]
24         public static void RawNumbersCRUDTest()
25         {
26             Using<byte>(links => links.TestRawNumbersCRUDOperations());
27             Using<ushort>(links => links.TestRawNumbersCRUDOperations());
28             Using<uint>(links => links.TestRawNumbersCRUDOperations());
29             Using<ulong>(links => links.TestRawNumbersCRUDOperations());
30         }
31
32         [Fact]
33         public static void MultipleRandomCreationsAndDeletionsTest()
34         {
35             Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
36                 ↳ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
37                 ↳ implementation of tree cuts out 5 bits from the address space.
38             Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
39                 ↳ stMultipleRandomCreationsAndDeletions(100));
40             Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
41                 ↳ MultipleRandomCreationsAndDeletions(100));
42             Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
43                 ↳ tMultipleRandomCreationsAndDeletions(100));
44         }
45         private static void Using<TLinkAddress>(Action<ILinks<TLinkAddress>> action) where
46             ↳ TLinkAddress : struct
47         {
48             var unitedMemoryLinks = new UnitedMemoryLinks<TLinkAddress>(new
49                 ↳ HeapResizableDirectMemory());
50             using (var logFile = File.Open("linksLogger.txt", FileMode.Create, FileAccess.Write))
51             {
52                 LoggingDecorator<TLinkAddress> links = new(unitedMemoryLinks, logFile);
53                 action(links);
54             }
55
56             File.Delete("db.links");
57             using var ffiLinks = new FFI.UnitedMemoryLinks<TLinkAddress>("db.links");
58             action(ffiLinks);
59         }
60     }
61 }

```

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

```

1  using System.IO;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Memory;
4  using Xunit;
5
6
7  namespace Platform.Data.Doublets.Tests
8  {

```

```

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

```

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

```

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

```

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

```

1  using System.IO;
2  using Xunit;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.United.Specific;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public static class ResizableDirectMemoryLinksTests
10     {

```

```

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

```

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

```

1  using Xunit;
2  using Platform.Scopes;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Reflection;
6  using Platform.Data.Doublets.Memory.United.Generic;
7  using Platform.Data.Doublets.Memory.United.Specific;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class ScopeTests
12     {
13         [Fact]
14         public static void SingleDependencyTest()
15         {
16             using (var scope = new Scope())
17             {
18                 scope.IncludeAssemblyOf<IMemory>();
19                 var instance = scope.Use<IDirectMemory>();

```

```

20         Assert.IsType<HeapResizableDirectMemory>(instance);
21     }
22 }
23
24 [Fact]
25 public static void CascadeDependencyTest()
26 {
27     using (var scope = new Scope())
28     {
29         scope.Include<TemporaryFileMappedResizableDirectMemory>();
30         scope.Include<UInt64UnitedMemoryLinks>();
31         var instance = scope.Use<ILinks<ulong>>();
32         Assert.IsType<UInt64UnitedMemoryLinks>(instance);
33     }
34 }
35
36 [Fact(Skip = "Would be fixed later.")]
37 public static void FullAutoResolutionTest()
38 {
39     using (var scope = new Scope(autoInclude: true, autoExplore: true))
40     {
41         var instance = scope.Use<UInt64Links>();
42         Assert.IsType<UInt64Links>(instance);
43     }
44 }
45
46 [Fact]
47 public static void TypeParametersTest()
48 {
49     using (var scope = new Scope<Types<HeapResizableDirectMemory,
50 ↪ UnitedMemoryLinks<ulong>>>())
51     {
52         var links = scope.Use<ILinks<ulong>>();
53         Assert.IsType<UnitedMemoryLinks<ulong>>(links);
54     }
55 }
56 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Generic;
5  using Platform.Data.Doublets.Memory;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryGenericLinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using<byte>(links => links.TestCRUDOperations());
15             Using<ushort>(links => links.TestCRUDOperations());
16             Using<uint>(links => links.TestCRUDOperations());
17             Using<ulong>(links => links.TestCRUDOperations());
18         }
19
20         [Fact]
21         public static void RawNumbersCRUDTest()
22         {
23             UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
24             UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
25             UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
26             UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
27         }
28
29         [Fact]
30         public static void MultipleRandomCreationsAndDeletionsTest()
31         {
32             Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
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));

```



```

35         Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
36     }
37     private static void Using<TLinkAddress>(Action<ILinks<TLinkAddress>> action) where
38     ↪ TLinkAddress : struct
39     {
40         using (var dataMemory = new HeapResizableDirectMemory())
41         using (var indexMemory = new HeapResizableDirectMemory())
42         using (var memory = new SplitMemoryLinks<TLinkAddress>(dataMemory, indexMemory))
43         {
44             action(memory);
45         }
46     }
47     private static void
48     ↪ UsingWithExternalReferences<TLinkAddress>(Action<ILinks<TLinkAddress>> action) where
49     ↪ TLinkAddress : struct
50     {
51         var constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
52         ↪ true);
53         using (var dataMemory = new HeapResizableDirectMemory())
54         using (var indexMemory = new HeapResizableDirectMemory())
55         using (var memory = new SplitMemoryLinks<TLinkAddress>(dataMemory, indexMemory,
56         ↪ SplitMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, constants))
57     {
58         action(memory);
59     }
60 }

```

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 TLinkAddress = 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<TLinkAddress>> 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<TLinkAddress>> action)
38         {
39             var constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
40             ↪ true);
41             using (var dataMemory = new HeapResizableDirectMemory())
42             using (var indexMemory = new HeapResizableDirectMemory())
43             using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory,
44             ↪ UInt32SplitMemoryLinks.DefaultLinksSizeStep, constants))
45         {
46             action(memory);
47         }
48     }
49 }

```

```

45     }
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 TLinkAddress = 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<TLinkAddress>> 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         private static void UsingWithExternalReferences(Action<ILinks<TLinkAddress>> action)
38         {
39             var constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
40                 true);
41             using (var dataMemory = new HeapResizableDirectMemory())
42             using (var indexMemory = new HeapResizableDirectMemory())
43             using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
44                 UInt64SplitMemoryLinks.DefaultLinksSizeStep, constants))
45             {
46                 action(memory);
47             }
48         }
49     }
50 }

```

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

```

1  using System.Collections.Generic;
2  using Xunit;
3  using Platform.Ranges;
4  using Platform.Numbers;
5  using Platform.Random;
6  using Platform.Setters;
7  using Platform.Converters;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class TestExtensions
12     {
13         public static void TestCRUDOperations<T>(this ILinks<T> links) where T : struct
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) where T : struct
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<TLinkAddress>(this
↪ ILinks<TLinkAddress> links, int numberOfOperations)
158 {
159     for (int i = 0; i < numberOfOperations; i++)
160     {
161         links.Create();
162     }
163     for (int i = 0; i < numberOfOperations; i++)
164     {
165         links.Delete(links.Count());
166     }
167 }
168
169 public static void TestMultipleRandomCreationsAndDeletions<TLinkAddress>(this
↪ ILinks<TLinkAddress> links, int maximumOperationsPerCycle) where TLinkAddress :
↪ struct
170 {
171     var comparer = Comparer<TLinkAddress>.Default;
172     var addressToUInt64Converter = CheckedConverter<TLinkAddress, ulong>.Default;
173     var uint64ToAddressConverter = CheckedConverter<ulong, TLinkAddress>.Default;
174     for (var N = 1; N < maximumOperationsPerCycle; N++)
175     {
176         var random = new System.Random(N);
177         var created = 0UL;

```

```

178     var deleted = 0UL;
179     for (var i = 0; i < N; i++)
180     {
181         var linksCount = addressToUInt64Converter.Convert(links.Count());
182         var createPoint = random.NextBoolean();
183         if (linksCount >= 2 && createPoint)
184         {
185             var linksAddressRange = new Range<ulong>(1, linksCount);
186             TLinkAddress source = uInt64ToAddressConverter.Convert(random.NextUInt64
187                 ↪ (linksAddressRange));
188             TLinkAddress target = uInt64ToAddressConverter.Convert(random.NextUInt64
189                 ↪ (linksAddressRange));
190             ↪ //-V3086
191             var resultLink = links.GetOrCreate(source, target);
192             if (comparer.Compare(resultLink,
193                 ↪ uInt64ToAddressConverter.Convert(linksCount)) > 0)
194             {
195                 created++;
196             }
197         }
198         else
199         {
200             links.Create();
201             created++;
202         }
203     }
204     Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
205     for (var i = 0; i < N; i++)
206     {
207         TLinkAddress link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
208         if (links.Exists(link))
209         {
210             links.Delete(link);
211             deleted++;
212         }
213     }
214     Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }

```

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 TLinkAddress = System.UInt64;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     public class UInt64LinksExtensionsTests
13     {
14         public static ILinks<TLinkAddress> CreateLinks() => CreateLinks<TLinkAddress>(new
15             ↪ Platform.IO.TemporaryFile());
16
17         public static ILinks<TLinkAddress> CreateLinks<TLinkAddress>(string dataDBFilename)
18             ↪ where TLinkAddress : struct
19         {
20             var linksConstants = new
21                 ↪ LinksConstants<TLinkAddress>(enableExternalReferencesSupport: true);
22             return new UnitedMemoryLinks<TLinkAddress>(new
23                 ↪ FileMappedResizableDirectMemory(dataDBFilename),
24                 ↪ UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
25                 ↪ IndexTreeType.Default);
26         }
27
28         [Fact]
29         public void FormatStructureWithExternalReferenceTest()
30         {
31             ILinks<TLinkAddress> links = CreateLinks();
32             TLinkAddress zero = default;
33             var one = Arithmetic.Increment(zero);
34             var markerIndex = one;
35             var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
36             var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
37                 ↪ markerIndex));
38         }
39     }
40 }

```

```

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

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLinkAddress = 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().TestMultipleRandomCreationsAndDeletions(100));
29         }
30         private static void Using(Action<ILinks<TLinkAddress>> action)
31         {
32             using (var scope = new Scope<Types<HeapResizableDirectMemory,
33                 ↪ UInt32UnitedMemoryLinks>>())
34             {
35                 action(scope.Use<ILinks<TLinkAddress>>());
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 TLinkAddress = 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().TestMultip
            ↳ leRandomCreationsAndDeletions(100));
29     }
30     private static void Using(Action<ILinks<TLinkAddress>> action)
31     {
32         using (var scope = new Scope<Types<HeapResizableDirectMemory,
            ↳ UInt64UnitedMemoryLinks>>())
33         {
34             action(scope.Use<ILinks<TLinkAddress>>());
35         }
36     }
37 }
38 }

```

Index

`./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.cs`, 461
`./csharp/Platform.Data.Doublets.Tests/ILinksBasicTests.cs`, 461
`./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs`, 462
`./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs`, 462
`./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs`, 463
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs`, 464
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs`, 465
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs`, 466
`./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs`, 466
`./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs`, 469
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs`, 470
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt64LinksTests.cs`, 470
`./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`, 13
`./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`, 18
`./csharp/Platform.Data.Doublets/Decorators/UniLinks.cs`, 20
`./csharp/Platform.Data.Doublets/Doublet.cs`, 26
`./csharp/Platform.Data.Doublets/DoubletComparer.cs`, 28
`./csharp/Platform.Data.Doublets/FFI/UInt32UnitedMemoryLinks.cs`, 28
`./csharp/Platform.Data.Doublets/FFI/UnitedMemoryLinks.cs`, 30
`./csharp/Platform.Data.Doublets/ILinks.cs`, 39
`./csharp/Platform.Data.Doublets/ILinksExtensions.cs`, 40
`./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs`, 60
`./csharp/Platform.Data.Doublets/Link.cs`, 61
`./csharp/Platform.Data.Doublets/LinkExtensions.cs`, 68
`./csharp/Platform.Data.Doublets/LinksOperatorBase.cs`, 69
`./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs`, 70
`./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs`, 70
`./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs`, 72
`./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs`, 72
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 74
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs`, 81
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 88
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 92
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 96
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 100
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 103
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs`, 109
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesLinkedListMethods.cs`, 115
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 120
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs`, 123
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 127
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs`, 131
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs`, 134
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs`, 138
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs`, 156
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs`, 159
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs`, 160
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 162
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase.cs`, 168
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 174
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 178

./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 182
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs, 186
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 190
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 196
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 202
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 203
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 206
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 210
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 214
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 218
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs, 224
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 225
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase.cs, 231
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 237
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods.cs, 241
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 245
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethods.cs, 249
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 253
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.cs, 258
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 264
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 265
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 269
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 273
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 276
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 280
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 286
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 287
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 297
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 303
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 310
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 315
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 319
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 323
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 328
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 332
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs, 335
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs, 338
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs, 351
./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs, 354
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 356
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 362
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 367
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs, 371
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 375
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 378
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 382
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 388
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 389
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 396
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 401
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 407
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 412
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 416
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 420
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 425
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 429
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 433
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs, 438
./csharp/Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs, 439
./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs, 441
./csharp/Platform.Data.Doublets/Stacks/Stack.cs, 442
./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs, 444

./csharp/Platform.Data.Doublets/SynchronizedLinks.cs, 444
./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs, 447
./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 451