

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>
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        /// <para></para>
33        /// </param>
34        /// <param name="targetToMatch">
35        /// <para>A target to match.</para>
36        /// <para></para>
37        /// </param>
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        public TargetMatcher(ILinks<TLinkAddress> links, TLinkAddress targetToMatch) :
40        ↪ base(links) => _targetToMatch = targetToMatch;
41
42        /// <summary>
43        /// <para>
44        /// Determines whether this instance is matched.
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>The bool</para>
54        /// <para></para>
55        /// </returns>
56        [MethodImpl(MethodImplOptions.AggressiveInlining)]
57        public bool IsMatched(TLinkAddress link) =>
58        ↪ _equalityComparer.Equals(_links.GetTarget(link), _targetToMatch);
59    }
60 }
```

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>
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>
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 = _links.GetIndex(restriction);
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;

```

```

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(IList<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(IList<TLinkAddress>? restriction,
        ↪ ReadHandler<TLinkAddress>? handler) => _links.Each(restriction, handler);

```

```

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 }
179 }

```

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
20     {

```

```

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>
97    /// <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>
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(IList<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 }

```

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links itself constant to self reference resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLinkAddress}"/>
17     public class LinksItselfConstantToSelfReferenceResolver<TLinkAddress> :
18     ↪  LinksDecoratorBase<TLinkAddress>
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.
26         /// </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>
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(links.GetSource(substitution), links.GetTarget(substitution));
58             return links.Update(restriction, substitution, handler);
59         }
60     }
61 }

```

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>
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="LinksNullConstantToSelfReferenceResolver"/> instance.
23         /// </para>
24         /// <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(IList<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(IList<TLinkAddress>? restriction,
        ↪ IList<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>
18     {
19         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
            ↪ EqualityComparer<TLinkAddress>.Default;
20

```

```

21     /// <summary>
22     /// <para>
23     /// Initializes a new <see cref="LinksUniquenessResolver"/> 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 LinksUniquenessResolver(ILinks<TLinkAddress> links) : base(links) { }
33
34     /// <summary>
35     /// <para>
36     /// Updates the restriction.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     /// <param name="restriction">
41     /// <para>The restriction.</para>
42     /// <para></para>
43     /// </param>
44     /// <param name="substitution">
45     /// <para>The substitution.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public override TLinkAddress Update(IList<TLinkAddress>? restriction,
54     ↪  IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
55     {
56         var constants = _constants;
57         var links = _links;
58         var newLinkAddress = links.SearchOrDefault(links.GetSource(substitution),
59         ↪  links.GetTarget(substitution));
60         if (_equalityComparer.Equals(newLinkAddress, default))
61         {
62             return links.Update(restriction, substitution, handler);
63         }
64         return ResolveAddressChangeConflict(links.GetIndex(restriction), newLinkAddress,
65         ↪  handler);
66     }
67
68     /// <summary>
69     /// <para>
70     /// Resolves the address change conflict using the specified old link address.
71     /// </para>
72     /// <para></para>
73     /// </summary>
74     /// <param name="oldLinkAddress">
75     /// <para>The old link address.</para>
76     /// <para></para>
77     /// </param>
78     /// <param name="newLinkAddress">
79     /// <para>The new link address.</para>
80     /// <para></para>
81     /// </param>
82     /// <returns>
83     /// <para>The new link address.</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected virtual TLinkAddress ResolveAddressChangeConflict(TLinkAddress oldLinkAddress,
88     ↪  TLinkAddress newLinkAddress, WriteHandler<TLinkAddress>? handler)
89     {
90         if (!_equalityComparer.Equals(oldLinkAddress, newLinkAddress) &&
91         ↪  _links.Exists(oldLinkAddress))
92         {
93             return _facade.Delete(oldLinkAddress, handler);
94         }
95         return _links.Constants.Continue;
96     }
97 }

```

```
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>
18     {
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="LinksUniquenessValidator"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="links">
26         /// <para>A links.</para>
27         /// <para></para>
28         /// </param>
29         [MethodImpl(MethodImplOptions.AggressiveInlining)]
30         public LinksUniquenessValidator(ILinks<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(ICollection<TLinkAddress>? restriction,
52             ↳ ICollection<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
53         {
54             var links = _links;
55             var constants = _constants;
56             links.EnsureDoesNotExists(links.GetSource(substitution),
57                 ↳ links.GetTarget(substitution));
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>
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,
52      ↪  IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
53      {
54          var links = _links;
55          links.EnsureNoUsages(links.GetIndex(restriction));
56          return links.Update(restriction, substitution, handler);
57      }
58
59      /// <summary>
60      /// <para>
61      /// Deletes the restriction.
62      /// </para>
63      /// <para></para>
64      /// </summary>
65      /// <param name="restriction">
66      /// <para>The restriction.</para>
67      /// <para></para>
68      /// </param>
69      [MethodImpl(MethodImplOptions.AggressiveInlining)]
70      public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
71      ↪  WriteHandler<TLinkAddress>? handler)
72      {
73          var links = _links;
74          var link = links.GetIndex(restriction);
75          links.EnsureNoUsages(link);
76          return links.Delete(restriction, handler);
77      }
78  }

```

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>
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,
19     ↪ WriteHandler<TLinkAddress>? handler)
20     {
21         WriteHandlerState<TLinkAddress> handlerState = new(_constants.Continue,
22         ↪ _constants.Break, handler);
23         return base.Create(substitution, (before, after) =>
24         {
25             handlerState.Handle(before, after);
26             _logStreamWriter.WriteLine($"Create. Before: {new Link<TLinkAddress>(before)}.
27             ↪ After: {new Link<TLinkAddress>(after)}");
28             return _constants.Continue;
29         });
30     }
31
32     public override TLinkAddress Update(IList<TLinkAddress>? restriction,
33     ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
34     {
35         WriteHandlerState<TLinkAddress> handlerState = new(_constants.Continue,
36         ↪ _constants.Break, handler);
37         return base.Update(restriction, substitution, (before, after) =>
38         {
39             handlerState.Handle(before, after);
40             _logStreamWriter.WriteLine($"Update. Before: {new Link<TLinkAddress>(before)}.
41             ↪ After: {new Link<TLinkAddress>(after)}");
42             return _constants.Continue;
43         });
44     }
45
46     public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
47     ↪ WriteHandler<TLinkAddress>? handler)
48     {
49         WriteHandlerState<TLinkAddress> handlerState = new(_constants.Continue,
50         ↪ _constants.Break, handler);
51         return base.Delete(restriction, (before, after) =>
52         {
53             handlerState.Handle(before, after);
54             _logStreamWriter.WriteLine($"Delete. Before: {new Link<TLinkAddress>(before)}.
55             ↪ After: {new Link<TLinkAddress>(after)}");
56             return _constants.Continue;
57         });
58     }
59 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using Platform.Delegates;
4  using Platform.Exceptions;
5
6  namespace Platform.Data.Doublets.Decorators;
7
8  public class NoExceptionsDecorator<TLinkAddress> : LinksDecoratorBase<TLinkAddress>
9  {
10     public NoExceptionsDecorator(ILinks<TLinkAddress> storage) : base(storage) { }
11
12     public override TLinkAddress Count(IList<TLinkAddress>? restriction)
13     {
14         try
15         {
16             return base.Count(restriction);
17         }
18         catch (Exception exception)
19         {
20             exception.Ignore();
21             return Constants.Error;
22         }
23     }
24
25     public override TLinkAddress Each(IList<TLinkAddress>? restriction,
26     ↪ ReadHandler<TLinkAddress>? handler)
27     {

```

```

27     try
28     {
29         return base.Each(restriction, handler);
30     }
31     catch (Exception exception)
32     {
33         exception.Ignore();
34         return Constants.Error;
35     }
36 }
37
38 public override TLinkAddress Create(IList<TLinkAddress>? substitution,
↪ WriteHandler<TLinkAddress>? handler)
39 {
40     try
41     {
42         return base.Create(substitution, handler);
43     }
44     catch (Exception exception)
45     {
46         exception.Ignore();
47         return Constants.Error;
48     }
49 }
50
51 public override TLinkAddress Update(IList<TLinkAddress>? restriction, IList<TLinkAddress>?
↪ substitution, WriteHandler<TLinkAddress>? handler)
52 {
53     try
54     {
55         return base.Update(restriction, substitution, handler);
56     }
57     catch (Exception exception)
58     {
59         exception.Ignore();
60         return Constants.Error;
61     }
62 }
63
64 public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
↪ WriteHandler<TLinkAddress>? handler)
65 {
66     try
67     {
68         return base.Delete(restriction, handler);
69     }
70     catch (Exception exception)
71     {
72         exception.Ignore();
73         return Constants.Error;
74     }
75 }
76 }

```

1.15 ./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>
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 = _links.GetIndex(restriction);
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 }

```

1.16 ./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,
49     ↪ WriteHandler<TLinkAddress>? handler) => _links.CreatePoint(handler);
50
51     /// <summary>
52     /// <para>
53     /// Updates the substitution.
54     /// </para>
55     /// <para></para>
56     /// </summary>
57     /// <param name="restriction">
58     /// <para>The substitution.</para>
59     /// <para></para>
60     /// </param>
61     /// <param name="substitution">
62     /// <para>The substitution.</para>
63     /// <para></para>
64     /// </param>
65     /// <returns>
66     /// <para>The link</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     public override TLinkAddress Update(ICollection<TLinkAddress>? restriction,
71     ↪ ICollection<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
72     {
73         var constants = _constants;
74         var indexPartConstant = constants.IndexPart;
75         var sourcePartConstant = constants.SourcePart;
76         var targetPartConstant = constants.TargetPart;
77         var nullConstant = constants.Null;
78         var itselfConstant = constants.Itself;
79         var existedLink = nullConstant;
80         var updatedLink = restriction[indexPartConstant];
81         var newSource = substitution[sourcePartConstant];
82         var newTarget = substitution[targetPartConstant];
83         var links = _links;
84         if (newSource != itselfConstant && newTarget != itselfConstant)
85         {
86             existedLink = links.SearchOrDefault(newSource, newTarget);
87         }
88         if (existedLink == nullConstant)
89         {
90             var before = links.GetLink(updatedLink);
91             if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
92             ↪ newTarget)
93             {
94                 var source = newSource == itselfConstant ? updatedLink : newSource;
95                 var target = newTarget == itselfConstant ? updatedLink : newTarget;
96                 return links.Update(new Link<TLinkAddress>(updatedLink, source, target),
97                 ↪ handler);
98             }
99             return _links.Constants.Continue;
100         }
101         else
102         {
103             return _facade.MergeAndDelete(updatedLink, existedLink, handler);
104         }
105     }
106
107     /// <summary>
108     /// <para>
109     /// Deletes the substitution.
110     /// </para>
111     /// <para></para>
112     /// </summary>
113     /// <param name="restriction">
114     /// <para>The substitution.</para>
115     /// <para></para>
116     /// </param>
117     [MethodImpl(MethodImplOptions.AggressiveInlining)]
118     public override TLinkAddress Delete(ICollection<TLinkAddress>? restriction,
119     ↪ WriteHandler<TLinkAddress>? handler)
120     {
121         var linkIndex = _links.GetIndex(restriction);
122         var constants = _links.Constants;
123         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
124         ↪ 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 }

```

1.17 ./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>
65         /// <para></para>
66         /// </returns>
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         public override TLinkAddress Create(ICollection<TLinkAddress>? substitution,
69         ↪ WriteHandler<TLinkAddress>? handler) => _links.CreatePoint(handler);
70     }
71 }

```

```

61
62     /// <summary>
63     /// <para>
64     /// Updates the substitution.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="restriction">
69     /// <para>The substitution.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="substitution">
73     /// <para>The substitution.</para>
74     /// <para></para>
75     /// </param>
76     /// <returns>
77     /// <para>The TLinkAddress</para>
78     /// <para></para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public override TLinkAddress Update(ICollection<TLinkAddress>? restriction,
82     ↪ ICollection<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
83     {
84         var constants = _constants;
85         var indexPartConstant = constants.IndexPart;
86         var sourcePartConstant = constants.SourcePart;
87         var targetPartConstant = constants.TargetPart;
88         var nullConstant = constants.Null;
89         var itselfConstant = constants.Itself;
90         var existedLink = nullConstant;
91         var updatedLink = restriction[indexPartConstant];
92         var newSource = substitution[sourcePartConstant];
93         var newTarget = substitution[targetPartConstant];
94         var links = _links;
95         if (newSource != itselfConstant && newTarget != itselfConstant)
96         {
97             existedLink = links.SearchOrDefault(newSource, newTarget);
98         }
99         if (existedLink == nullConstant)
100         {
101             var before = links.GetLink(updatedLink);
102             if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
103             ↪ newTarget)
104             {
105                 var source = newSource == itselfConstant ? updatedLink : newSource;
106                 var target = newTarget == itselfConstant ? updatedLink : newTarget;
107                 return links.Update(new Link<TLinkAddress>(updatedLink, source, target),
108                 ↪ handler);
109             }
110             return _links.Constants.Continue;
111         }
112         else
113         {
114             return _facade.MergeAndDelete(updatedLink, existedLink, handler);
115         }
116     }
117
118     /// <summary>
119     /// <para>
120     /// Deletes the substitution.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="restriction">
125     /// <para>The substitution.</para>
126     /// <para></para>
127     /// </param>
128     [MethodImpl(MethodImplOptions.AggressiveInlining)]
129     public override TLinkAddress Delete(ICollection<TLinkAddress>? restriction,
130     ↪ WriteHandler<TLinkAddress>? handler)
131     {
132         var linkIndex = _links.GetIndex(restriction);
133         var constants = _links.Constants;
134         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
135         ↪ constants.Break, handler);
136         handlerState.Apply(_links.EnforceResetValues(linkIndex, handlerState.Handler));
137         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
138         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));

```

```

134         return handlerState.Result;
135     }
136 }
137 }

```

1.18 ./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>
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>
70             public Transition(IList<TLinkAddress>? before, IList<TLinkAddress>? after)
71             {
72                 Before = before;
73                 After = after;
74             }
75         }
76     }
77 }

```

```

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     ///                    newValue[Constants.SourcePart] =
142     ///                    ↳ Equals(substitution[Constants.SourcePart], Constants.Itself) ?
143     ///                    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.SourcePart];

```

```

134      newLink[Constants.TargetPart] =
135      ↪ Equals(substitution[Constants.TargetPart], Constants.Itself) ?
136      ↪ matchedLink[Constants.IndexPart] : substitution[Constants.TargetPart];
137      var matchDecision = matchedHandler(matchedLink, newValue);
138      if (Equals(matchDecision, Constants.Break))
139      ↪ return false;
140      if (!Equals(matchDecision, Constants.Skip))
141      ↪ transitions.Add(new Transition(matchedLink, newValue));
142      return true;
143      };
144      if (!Memory.Each(handler, restriction))
145      ↪ return Constants.Break;
146      }
147      }
148      else
149      {
150      Func<T, bool> handler = link =>
151      {
152      var matchedLink = Memory.GetLinkValue(link);
153      var matchDecision = matchedHandler(matchedLink, matchedLink);
154      return !Equals(matchDecision, Constants.Break);
155      };
156      if (!Memory.Each(handler, restriction))
157      ↪ return Constants.Break;
158      }
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      }
182      }
183      }
184      }
185      }
186      }
187      }
188      }
189      }
190      }
191      }
192      }
193      }
194      }
195      }
196      }
197      }
198      }
199      }
200      }
201      }
202      }
203      }
204      }
205      }
206      }
207      }

```

```

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 return _constants.Continue;
256 }
257
258 /// <summary>
259 /// <para>
260 /// Triggers the pattern or condition.
261 /// </para>
262 /// <para></para>
263 /// </summary>
264 /// <param name="patternOrCondition">
265 /// <para>The pattern or condition.</para>
266 /// <para></para>
267 /// </param>
268 /// <param name="matchHandler">
269 /// <para>The match handler.</para>
270 /// <para></para>
271 /// </param>
272 /// <param name="substitution">
273 /// <para>The substitution.</para>
274 /// <para></para>
275 /// </param>
276 /// <param name="substitutionHandler">
277 /// <para>The substitution handler.</para>
278 /// <para></para>
279 /// </param>
280 /// <exception cref="NotImplementedException">
281 /// <para></para>
282 /// </exception>
283 /// <exception cref="NotSupportedException">
284 /// <para></para>
285 /// </exception>

```



```

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:
309     ↪ Check if it is a correct condition
310     {
311         // Or it only applies to trigger without matchHandler.
312         throw new NotImplementedException();
313     }
314     else if (!substitution.IsNullOrEmpty()) // Creation
315     {
316         var before = Array.Empty<TLinkAddress>();
317         // Что должно означать False здесь? Остановиться (перестать идти) или пропустить
318         ↪ (пройти мимо) или пустить (взять)?
319         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
320             ↪ constants.Break))
321         {
322             return constants.Break;
323         }
324         var after = (IList<TLinkAddress>?)substitution.ToArray();
325         if (_equalityComparer.Equals(after[0], default))
326         {
327             var newLink = _links.Create();
328             after[0] = newLink;
329         }
330         if (substitution.Count == 1)
331         {
332             after = _links.GetLink(substitution[0]);
333         }
334         else if (substitution.Count == 3)
335         {
336             //Links.Create(after);
337         }
338         else
339         {
340             throw new NotSupportedException();
341         }
342         return matchHandler != null ? substitutionHandler(before, after) :
343             ↪ constants.Continue;
344     }
345     else if (!patternOrCondition.IsNullOrEmpty()) // Deletion
346     {
347         if (patternOrCondition.Count == 1)
348         {
349             var linkToDelete = patternOrCondition[0];
350             var before = _links.GetLink(linkToDelete);
351             if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
352                 ↪ constants.Break))
353             {
354                 return constants.Break;
355             }
356             var after = Array.Empty<TLinkAddress>();
357             _links.Update(linkToDelete, constants.Null, constants.Null);
358             _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;
}

private TLinkAddress AlwaysContinue(IList<TLinkAddress>? linkToMatch) =>
    ↪ _constants.Continue;

```

1.19 ./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 =
21             ↳ EqualityComparer<T>.Default;
22
23         /// <summary>
24         /// <para>.</para>
25         /// <para>.</para>
26         /// </summary>
27         /// <typeparam name="T">
28         /// <para>.</para>
29         /// <para>.</para>
30         /// </typeparam>
31         public readonly T Source;
32
33         /// <summary>
34         /// <para>.</para>
35         /// <para>.</para>
36         /// </summary>
37         /// <typeparam name="T">
38         /// <para>.</para>
39         /// <para>.</para>
40         /// </typeparam>
41         public readonly T Target;
42
43         /// <summary>
44         /// <para>.</para>
45         /// <para>.</para>
46         /// </summary>
47         /// <typeparam name="T">
48         /// <para>.</para>
49         /// <para>.</para>
50         /// </typeparam>
51         /// <param name="source">
52         /// <para>.</para>
53         /// <para>.</para>
54         /// </param>
55         /// <param name="target">
56         /// <para>.</para>
57         /// <para>.</para>
58         /// </param>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public Doublet(T source, T target)
61         {
62             Source = source;
63             Target = target;
64         }
65
66         /// <summary>
67         /// <para>.</para>
68         /// <para>.</para>
69         /// </summary>
70         /// <returns>
71         /// <para>.</para>
72         /// <para>.</para>
73         /// </returns>
74         [MethodImpl(MethodImplOptions.AggressiveInlining)]
75         public override string ToString() => $"{Source}->{Target}";
76
77         /// <summary>
78         /// <para>.</para>

```

```

78     /// <para>.</para>
79     /// </summury>
80     /// <typeparam>
81     /// <para>.</para>
82     /// <para>.</para>
83     /// </typeparam>
84     /// <param name="other">
85     /// <para>.</para>
86     /// <para>.</para>
87     /// </param>
88     /// <returns>
89     /// <para>.</para>
90     /// <para>.</para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public bool Equals(Doublet<T> other) => _equalityComparer.Equals(Source, other.Source)
94     ↪ && _equalityComparer.Equals(Target, other.Target);
95
96     /// <summury>
97     /// <para>.</para>
98     /// <para>.</para>
99     /// </summury>
100    /// <typeparam>
101    /// <para>.</para>
102    /// <para>.</para>
103    /// </typeparam>
104    /// <param name="obj">
105    /// <para>.</para>
106    /// <para>.</para>
107    /// </param>
108    /// <returns>
109    /// <para>.</para>
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    /// <summury>
117    /// <para>.</para>
118    /// <para>.</para>
119    /// </summury>
120    /// <returns>
121    /// <para>.</para>
122    /// <para>.</para>
123    /// </returns>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    public override int GetHashCode() => (Source, Target).GetHashCode();
126
127    /// <summury>
128    /// <para>.</para>
129    /// <para>.</para>
130    /// </summury>
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    /// <summury>
147    /// <para>.</para>
148    /// <para>.</para>
149    /// </summury>
150    /// <param name="left">
151    /// <para>.</para>
152    /// <para>.</para>
153    /// </param>
154    /// <param name="right">
155    /// <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 }

```

1.20 ./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 }

```

1.21 ./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                 public TLinkAddress Each(IList<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
44                     ↪ handler)
45                 {
46                     unsafe
47                     {
48                         Methods.EachCallback_UInt32 callback = (link) => handler != null ? handler(new
49                             ↪ Link<TLinkAddress>(link.Index, link.Source, link.Target)) :
50                             ↪ Constants.Continue;
51                         var array = stackalloc uint[restriction.Count];
52                         for (var i = 0; i < restriction.Count; i++)
53                         {
54                             array[i] = restriction[i];
55                         }
56                         return Methods.UInt32UnitedMemoryLinks_Each(_ptr, array,
57                             ↪ (nuint)(restriction?.Count ?? 0), callback);
58                     }
59                 }
60
61                 public TLinkAddress Create(IList<TLinkAddress>? substitution,
62                     ↪ WriteHandler<TLinkAddress>? handler)
63                 {
64                     unsafe
65                     {
66                         Methods.CreateCallback_UInt32 callback = (before, after) => handler != null ?
67                             ↪ handler(new Link<TLinkAddress>(before.Index, before.Source, before.Target),
68                             ↪ new Link<TLinkAddress>(after.Index, after.Source, after.Target)) :
69                             ↪ Constants.Continue;
70                         fixed (uint* substitutionPtr = (uint[])substitution)
71                         {
72                             return Methods.UInt32UnitedMemoryLinks_Create(_ptr, substitutionPtr,
73                                 ↪ (nuint)(substitution?.Count ?? 0), callback);
74                         }
75                     }
76                 }
77
78                 public TLinkAddress Update(IList<TLinkAddress>? restriction, IList<TLinkAddress>?
79                     ↪ substitution, WriteHandler<TLinkAddress>? handler)
80                 {
81                     unsafe

```

```

72     {
73         var restrictionArray = stackalloc uint[restriction.Count];
74         for (var i = 0; i < restriction.Count; i++)
75         {
76             restrictionArray[i] = restriction[i];
77         }
78         var substitutionArray = stackalloc uint[substitution.Count];
79         for (var i = 0; i < restriction.Count; i++)
80         {
81             substitutionArray[i] = restriction[i];
82         }
83         Methods.UpdateCallback_UInt32 callback = (before, after) => handler != null ?
            ↪ handler(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,
            ↪ (nuint)(restriction?.Count ?? 0), substitutionArray,
            ↪ (nuint)(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 != null ?
            ↪ handler(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,
            ↪ (nuint)(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 }

```

1.22 ./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     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
101     public static extern void* ByteUnitedMemoryLinks_New(string path);
102
103     [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 ulong UInt64UnitedMemoryLinks_Create(void* self, ulong*
129     ↳ substitution, nuint substitutionLength, CreateCallback_UInt64 callback);
130
131 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
132 public static extern byte ByteUnitedMemoryLinks_Count(void* self, byte* restriction,
133     ↳ nuint len);
134
135 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
136 public static extern ushort UInt16UnitedMemoryLinks_Count(void* self, ushort*
137     ↳ restriction, nuint len);
138
139 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
140 public static extern uint UInt32UnitedMemoryLinks_Count(void* self, uint* restriction,
141     ↳ nuint len);
142
143 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
144 public static extern ulong UInt64UnitedMemoryLinks_Count(void* self, ulong* restriction,
145     ↳ nuint len);
146
147 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
148 public static extern byte ByteUnitedMemoryLinks_Each(void* self, byte* restriction,
149     ↳ nuint len, EachCallback_UInt8 callback);
150
151 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
152 public static extern ushort UInt16UnitedMemoryLinks_Each(void* self, ushort*
153     ↳ restriction, nuint len, EachCallback_UInt16 callback);
154
155 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
156 public static extern uint UInt32UnitedMemoryLinks_Each(void* self, uint* restriction,
157     ↳ nuint len, EachCallback_UInt32 callback);
158
159 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
160 public static extern ulong UInt64UnitedMemoryLinks_Each(void* self, ulong* restriction,
161     ↳ nuint len, EachCallback_UInt64 callback);
162
163 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
164 public static extern byte ByteUnitedMemoryLinks_Update(void* self, byte* restriction,
165     ↳ nuint restrictionLength, byte* substitution, nuint substitutionLength,
166     ↳ UpdateCallback_UInt8 callback);
167
168 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
169 public static extern ushort UInt16UnitedMemoryLinks_Update(void* self, ushort*
170     ↳ restriction, nuint restrictionLength, ushort* substitution, nuint
171     ↳ substitutionLength, UpdateCallback_UInt16 callback);

```

```

157 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
158 public static extern uint UInt32UnitedMemoryLinks_Update(void* self, uint* restriction,
    ↳ nuint restrictionLength, uint* substitution, nuint substitutionLength,
    ↳ UpdateCallback UInt32 callback);

159 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
160 public static extern ulong UInt64UnitedMemoryLinks_Update(void* self, ulong*
    ↳ restriction, nuint restrictionLength, ulong* substitution, nuint
    ↳ substitutionLength, UpdateCallback UInt64 callback);

162 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
163 public static extern byte ByteUnitedMemoryLinks_Delete(void* self, byte* restriction,
164     ↳ nuint restrictionLength, DeleteCallback UInt8 callback);

165 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
166 public static extern ushort UInt16UnitedMemoryLinks_Delete(void* self, ushort*
167     ↳ restriction, nuint len, DeleteCallback UInt16 callback);

168 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
169 public static extern uint UInt32UnitedMemoryLinks_Delete(void* self, uint* restriction,
170     ↳ nuint len, DeleteCallback UInt32 callback);

171 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
172 public static extern ulong UInt64UnitedMemoryLinks_Delete(void* self, ulong*
173     ↳ restriction, nuint len, DeleteCallback UInt64 callback);
174 }

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

187     public LinksConstants<TLinkAddress> Constants { get; }

188     private readonly unsafe void* _ptr;

189     public UnitedMemoryLinks(string path)
190     {
191         TLinkAddress t = default;
192         unsafe
193         {
194             _ptr = t switch
195             {
196                 byte => Methods.ByteUnitedMemoryLinks_New(path),
197                 ushort => Methods.UInt16UnitedMemoryLinks_New(path),
198                 uint => Methods.UInt32UnitedMemoryLinks_New(path),
199                 ulong => Methods.UInt64UnitedMemoryLinks_New(path),
200                 _ => throw new NotImplementedException()
201             };

202             // TODO: Update api
203             Constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
                ↳ true);
204         }
205     }

206     public TLinkAddress Count(ICollection<TLinkAddress>? restriction)
207     {
208         var restrictionLength = restriction?.Count ?? 0;
209         unsafe
210         {
211             TLinkAddress t = default;
212             switch (t)
213             {
214                 case byte:
215                 {
216                     var restrictionArray = stackalloc byte[restrictionLength];
217                     var byteRestrictionArray = (ICollection<byte>)restriction;
218                     for (var i = 0; i < restrictionLength; i++)
219                     {
220                         restrictionArray[i] = byteRestrictionArray[i];
221                     }
222                 }
223             }
224         }
225     }

```

```

222     }
223     return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Count(_ptr,
224                             ↪ restrictionArray, (nuint)restrictionLength));
225 }
226 case ushort:
227 {
228     var restrictionArray = stackalloc ushort[restrictionLength];
229     var ushortRestrictionArray = (IList<ushort>)restriction;
230     for (var i = 0; i < restrictionLength; i++)
231     {
232         restrictionArray[i] = ushortRestrictionArray[i];
233     }
234     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Count(_ptr,
235                             ↪ restrictionArray, (nuint)restrictionLength));
236 }
237 case uint:
238 {
239     var restrictionArray = stackalloc uint[restrictionLength];
240     var uintRestrictionArray = (IList<uint>)restriction;
241     for (var i = 0; i < restrictionLength; i++)
242     {
243         restrictionArray[i] = uintRestrictionArray[i];
244     }
245     return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Count(_ptr,
246                             ↪ restrictionArray, (nuint)restrictionLength));
247 }
248 case ulong:
249 {
250     {
251         var restrictionArray = stackalloc ulong[restrictionLength];
252         var ulongRestrictionArray = (IList<ulong>)restriction;
253         for (var i = 0; i < restrictionLength; i++)
254         {
255             restrictionArray[i] = ulongRestrictionArray[i];
256         }
257         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Count(_ptr,
258                                 ↪ restrictionArray, (nuint)restrictionLength));
259     }
260 }
261 default:
262 {
263     throw new NotImplementedException();
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

287     var restrictionArray = stackalloc ushort[restrictionLength];
288     var ushortRestrictionArray = (IList<ushort>)restriction;
289     for (var i = 0; i < restrictionLength; i++)
290     {
291         restrictionArray[i] = ushortRestrictionArray[i];
292     }
293     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Each(_ptr,
294         ↪ restrictionArray, (nuint)restrictionLength, Callback));
295 }
296 case uint:
297 {
298     uint Callback(FfiLink_UInt32 link) => (uint)from_t.Convert(handler !=
299         ↪ null ? handler(new Link<TLinkAddress>(from_u32.Convert(link.Index),
300         ↪ from_u32.Convert(link.Source), from_u32.Convert(link.Target))) :
301         ↪ Constants.Continue);
302     var restrictionArray = stackalloc uint[restrictionLength];
303     var uintRestrictionArray = (IList<uint>)restriction;
304     for (var i = 0; i < restrictionLength; i++)
305     {
306         restrictionArray[i] = uintRestrictionArray[i];
307     }
308     return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Each(_ptr,
309         ↪ restrictionArray, (nuint)restrictionLength, Callback));
310 }
311 case ulong:
312 {
313     {
314         ulong Callback(FfiLink_UInt64 link) => from_t.Convert(handler !=
315             ↪ null ? handler(new
316             ↪ Link<TLinkAddress>(from_u64.Convert(link.Index),
317             ↪ from_u64.Convert(link.Source), from_u64.Convert(link.Target))) :
318             ↪ Constants.Continue);
319         var restrictionArray = stackalloc UInt64[restrictionLength];
320         var ulongRestrictionArray = (IList<ulong>)restriction;
321         for (var i = 0; i < restrictionLength; i++)
322         {
323             restrictionArray[i] = ulongRestrictionArray[i];
324         }
325         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Each(_ptr,
326             ↪ restrictionArray, (nuint)restrictionLength, Callback));
327     }
328 }
329 default:
330 {
331     throw new NotImplementedException();
332 }
333 }
334 }
335 }
336 }
337 }
338
339 public TLinkAddress Create(IList<TLinkAddress>? substitution,
340     ↪ WriteHandler<TLinkAddress>? handler)
341 {
342     var substitutionLength = substitution?.Count ?? 0;
343     unsafe
344     {
345         TLinkAddress t = default;
346         switch (t)
347         {
348             case byte:
349             {
350                 byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
351                     (byte)from_t.Convert(handler != null ? handler(new
352                     ↪ Link<TLinkAddress>(from_u8.Convert(before.Index),
353                     ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
354                     ↪ Link<TLinkAddress>(from_u8.Convert(after.Index),
355                     ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
356                     ↪ Constants.Continue);
357                 var substitutionArray = stackalloc byte[substitutionLength];
358                 var byteSubstitutionArray = (IList<byte>)substitution;
359                 for (var i = 0; i < substitutionLength; i++)
360                 {
361                     substitutionArray[i] = byteSubstitutionArray[i];
362                 }
363                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Create(_ptr,
364                     ↪ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
365             }
366             case ushort:

```

```

347     {
348         ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
            ↪ (ushort)from_t.Convert(handler != null ? handler(new
            ↪ Link<TLinkAddress>(from_u16.Convert(before.Index),
            ↪ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
            ↪ new Link<TLinkAddress>(from_u16.Convert(after.Index),
            ↪ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
            ↪ Constants.Continue);
349     var substitutionArray = stackalloc ushort[substitutionLength];
350     var ushortSubstitutionArray = (IList<ushort>)substitution;
351     for (var i = 0; i < substitutionLength; i++)
352     {
353         substitutionArray[i] = ushortSubstitutionArray[i];
354     }
355     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Create(_ptr,
            ↪ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
356 }
357 case uint:
358 {
359     uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
            ↪ (uint)from_t.Convert(handler != null ? handler(new
            ↪ Link<TLinkAddress>(from_u32.Convert(before.Index),
            ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
            ↪ new Link<TLinkAddress>(from_u32.Convert(after.Index),
            ↪ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) :
            ↪ Constants.Continue);
360     var substitutionArray = stackalloc uint[substitutionLength];
361     var uintSubstitutionArray = (IList<uint>)substitution;
362     for (var i = 0; i < substitutionLength; i++)
363     {
364         substitutionArray[i] = uintSubstitutionArray[i];
365     }
366     return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Create(_ptr,
            ↪ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
367 }
368 case ulong:
369 {
370     ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
            ↪ (ulong)from_t.Convert(handler != null ? handler(new
            ↪ Link<TLinkAddress>(from_u64.Convert(before.Index),
            ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
            ↪ new Link<TLinkAddress>(from_u64.Convert(after.Index),
            ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
            ↪ Constants.Continue);
372     var substitutionArray = stackalloc ulong[substitutionLength];
373     var ulongSubstitutionArray = (IList<ulong>)substitution;
374     for (var i = 0; i < substitutionLength; i++)
375     {
376         substitutionArray[i] = ulongSubstitutionArray[i];
377     }
378     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Create(_ptr,
            ↪ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
379 }
380 default:
381 {
382     throw new NotImplementedException();
383 }
384 };
385 }
386 }
387 }
388
389 public TLinkAddress Update(IList<TLinkAddress>? restriction, IList<TLinkAddress>?
    ↪ substitution, WriteHandler<TLinkAddress>? handler)
390 {
391     var restrictionLength = restriction?.Count ?? 0;
392     var substitutionLength = substitution?.Count ?? 0;
393     unsafe
394     {
395         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 != null ? handler(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 != null ? handler(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 != null ? handler(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 case ulong:
474 {
475     var restrictionArray = stackalloc ulong[restrictionLength];
476     var ulongRestrictionArray = (IList<ulong>)restriction;
477     for (var i = 0; i < restrictionLength; i++)
478     {
479         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 != null ? handler(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 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

```

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

```

1.23 ./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 }

```

1.24 ./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)
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 =
60                     ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
61                 links.GetOrCreate(source, target);
62             }
63         }
64
65         /// <summary>
66         /// <para>
67         /// 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,
81     ↪     ulong amountOfSearches)
82     {
83         var random = RandomHelpers.Default;
84         var addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
85         var uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
86         for (var i = OUL; i < amountOfSearches; i++)
87         {
88             var linksAddressRange = new Range<ulong>(0,
89             ↪     addressToUInt64Converter.Convert(links.Count()));
90             var source =
91             ↪     uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
92             var target =
93             ↪     uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
94             links.SearchOrDefault(source, target);
95         }
96     }
97
98     /// <summary>
99     /// <para>
100     /// Runs the random deletions using the specified links.
101     /// </para>
102     /// <para></para>
103     /// </summary>
104     /// <typeparam name="TLinkAddress">
105     /// <para>The link.</para>
106     /// <para></para>
107     /// </typeparam>
108     /// <param name="links">
109     /// <para>The links.</para>
110     /// <para></para>
111     /// </param>
112     /// <param name="amountOfDeletions">
113     /// <para>The amount of deletions.</para>
114     /// <para></para>
115     /// </param>
116     [MethodImpl(MethodImplOptions.AggressiveInlining)]
117     public static void RunRandomDeletions<TLinkAddress>(this ILinks<TLinkAddress> links,
118     ↪     ulong amountOfDeletions)
119     {
120         var random = RandomHelpers.Default;
121         var addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
122         var uInt64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
123         var linksCount = addressToUInt64Converter.Convert(links.Count());
124         var min = amountOfDeletions > linksCount ? OUL : linksCount - amountOfDeletions;
125         for (var i = OUL; i < amountOfDeletions; i++)
126         {
127             linksCount = addressToUInt64Converter.Convert(links.Count());
128             if (linksCount <= min)
129             {
130                 break;
131             }
132             var linksAddressRange = new Range<ulong>(min, linksCount);
133             var link =
134             ↪     uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
135             links.Delete(link);
136         }
137     }
138
139     /// <summary>
140     /// <para>
141     /// 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,
153    ↪ TLinkAddress linkToDelete, WriteHandler<TLinkAddress>? handler)
154    {
155        if (links.Exists(linkToDelete))
156        {
157            links.EnforceResetValues(linkToDelete, handler);
158        }
159        return links.Delete(new LinkAddress<TLinkAddress>(linkToDelete), handler);
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 удалится 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)
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 =
173    ↪ Arithmetic.Decrement(i))
174        {
175            links.Delete(i);
176            if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
177            {
178                i = links.Count();
179            }
180        }
181    }
182    /// <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)
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(new Link<TLinkAddress>(links.Constants.Any, 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)
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
280 /// <remarks>
281 /// TODO: Как так? Как то что ниже может быть корректно?
282 /// Скорее всего практически не применимо
283 /// Предполагалось, что можно было конвертировать формируемый в проходе через
    ↪ SequenceWalker
284 /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
285 /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
286 /// </remarks>

```

```

287 [MethodImpl(MethodImplOptions.AggressiveInlining)]
288 public static bool CheckPathExistence<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ params TLinkAddress[] path)
289 {
290     var current = path[0];
291     //EnsureLinkExists(current, "path");
292     if (!links.Exists(current))
293     {
294         return false;
295     }
296     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
297     var constants = links.Constants;
298     for (var i = 1; i < path.Length; i++)
299     {
300         var next = path[i];
301         var values = links.GetLink(current);
302         var source = links.GetSource(values);
303         var target = links.GetTarget(values);
304         if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
    ↪ next))
305         {
306             //throw new InvalidOperationException(string.Format("Невозможно выбрать
    ↪ путь, так как и Source и Target совпадают с элементом пути {0}.", next));
            return false;
307         }
308         if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
    ↪ target))
309         {
310             //throw new InvalidOperationException(string.Format("Невозможно продолжить
    ↪ путь через элемент пути {0}", next));
            return false;
311         }
312     }
313     current = next;
314 }
315 return true;
316 }
317
318
319 /// <remarks>
320 /// Может потребовать дополнительного стека для PathElement's при использовании
    ↪ SequenceWalker.
321 /// </remarks>
322 [MethodImpl(MethodImplOptions.AggressiveInlining)]
323 public static TLinkAddress GetByKeyes<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ TLinkAddress root, params int[] path)
324 {
325     links.EnsureLinkExists(root, "root");
326     var currentLink = root;
327     for (var i = 0; i < path.Length; i++)
328     {
329         currentLink = links.GetLink(currentLink)[path[i]];
330     }
331     return currentLink;
332 }
333
334 /// <summary>
335 /// <para>
336 /// Gets the square matrix sequence element by index using the specified links.
337 /// </para>
338 /// <para></para>
339 /// </summary>
340 /// <typeparam name="TLinkAddress">
341 /// <para>The link.</para>
342 /// <para></para>
343 /// </typeparam>
344 /// <param name="links">
345 /// <para>The links.</para>
346 /// <para></para>
347 /// </param>
348 /// <param name="root">
349 /// <para>The root.</para>
350 /// <para></para>
351 /// </param>
352 /// <param name="size">
353 /// <para>The size.</para>
354 /// <para></para>
355 /// </param>
356 /// <param name="index">
357 /// <para>The index.</para>

```

```

358 /// <para></para>
359 /// </param>
360 /// <exception cref="ArgumentOutOfRangeException">
361 /// <para>Sequences with sizes other than powers of two are not supported.</para>
362 /// <para></para>
363 /// </exception>
364 /// <returns>
365 /// <para>The current link.</para>
366 /// <para></para>
367 /// </returns>
368 [MethodImpl(MethodImplOptions.AggressiveInlining)]
369 public static TLinkAddress GetSquareMatrixSequenceElementByIndex<TLinkAddress>(this
    ↳ ILinks<TLinkAddress> links, TLinkAddress root, ulong size, ulong index)
370 {
371     var constants = links.Constants;
372     var source = constants.SourcePart;
373     var target = constants.TargetPart;
374     if (!Platform.Numbers.Math.IsPowerOfTwo(size))
375     {
376         throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
            ↳ than powers of two are not supported.");
377     }
378     var path = new BitArray(BitConverter.GetBytes(index));
379     var length = Bit.GetLowestPosition(size);
380     links.EnsureLinkExists(root, "root");
381     var currentLink = root;
382     for (var i = length - 1; i >= 0; i--)
383     {
384         currentLink = links.GetLink(currentLink)[path[i] ? target : source];
385     }
386     return currentLink;
387 }
388
389 #endregion
390
391 /// <summary>
392 /// Возвращает индекс указанной связи.
393 /// </summary>
394 /// <param name="links">Хранилище связей.</param>
395 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
396 /// <returns>Индекс начальной связи для указанной связи.</returns>
397 [MethodImpl(MethodImplOptions.AggressiveInlining)]
398 public static TLinkAddress GetIndex<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ IList<TLinkAddress>? link) => link[links.Constants.IndexPart];
399
400 /// <summary>
401 /// Возвращает индекс начальной (Source) связи для указанной связи.
402 /// </summary>
403 /// <param name="links">Хранилище связей.</param>
404 /// <param name="link">Индекс связи.</param>
405 /// <returns>Индекс начальной связи для указанной связи.</returns>
406 [MethodImpl(MethodImplOptions.AggressiveInlining)]
407 public static TLinkAddress GetSource<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress link) => 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)]
416 public static TLinkAddress GetSource<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ IList<TLinkAddress>? link) => link[links.Constants.SourcePart];
417
418 /// <summary>
419 /// Возвращает индекс конечной (Target) связи для указанной связи.
420 /// </summary>
421 /// <param name="links">Хранилище связей.</param>
422 /// <param name="link">Индекс связи.</param>
423 /// <returns>Индекс конечной связи для указанной связи.</returns>
424 [MethodImpl(MethodImplOptions.AggressiveInlining)]
425 public static TLinkAddress GetTarget<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress link) => links.GetLink(link)[links.Constants.TargetPart];
426
427 /// <summary>

```

```

428     /// Возвращает индекс конечной (Target) связи для указанной связи.
429     /// </summary>
430     /// <param name="links">Хранилище связей.</param>
431     /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↪ содержимого.</param>
432     /// <returns>Индекс конечной связи для указанной связи.</returns>
433     [MethodImpl(MethodImplOptions.AggressiveInlining)]
434     public static TLinkAddress GetTarget<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ IList<TLinkAddress>? link) => link[links.Constants.TargetPart];
435
436     /// <summary>
437     /// <para>
438     /// Alls the links.
439     /// </para>
440     /// <para></para>
441     /// </summary>
442     /// <typeparam name="TLinkAddress">
443     /// <para>The link.</para>
444     /// <para></para>
445     /// </typeparam>
446     /// <param name="links">
447     /// <para>The links.</para>
448     /// <para></para>
449     /// </param>
450     /// <param name="restriction">
451     /// <para>The restriction.</para>
452     /// <para></para>
453     /// </param>
454     /// <returns>
455     /// <para>A list of i list t link</para>
456     /// <para></para>
457     /// </returns>
458     [MethodImpl(MethodImplOptions.AggressiveInlining)]
459     public static IList<IList<TLinkAddress>?> All<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, params TLinkAddress[] restriction)
460     {
461         var allLinks = new List<IList<TLinkAddress>?>();
462         var filler = new ListFiller<IList<TLinkAddress>?, TLinkAddress>(allLinks,
    ↪ links.Constants.Continue);
463         links.Each(filler.AddAndReturnConstant, restriction);
464         return allLinks;
465     }
466
467     /// <summary>
468     /// <para>
469     /// Alls the indices using the specified links.
470     /// </para>
471     /// <para></para>
472     /// </summary>
473     /// <typeparam name="TLinkAddress">
474     /// <para>The link.</para>
475     /// <para></para>
476     /// </typeparam>
477     /// <param name="links">
478     /// <para>The links.</para>
479     /// <para></para>
480     /// </param>
481     /// <param name="restriction">
482     /// <para>The restriction.</para>
483     /// <para></para>
484     /// </param>
485     /// <returns>
486     /// <para>A list of t link</para>
487     /// <para></para>
488     /// </returns>
489     [MethodImpl(MethodImplOptions.AggressiveInlining)]
490     public static IList<TLinkAddress>? AllIndices<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, params TLinkAddress[] restriction)
491     {
492         var allIndices = new List<TLinkAddress>();
493         var filler = new ListFiller<TLinkAddress, TLinkAddress>(allIndices,
    ↪ links.Constants.Continue);
494         links.Each(filler.AddFirstAndReturnConstant, restriction);
495         return allIndices;
496     }
497
498     /// <summary>

```

```

/// Возвращает значение, определяющее существует ли связь с указанными началом и концом
↪ в хранилище связей.
/// </summary>
/// <param name="links">Хранилище связей.</param>
/// <param name="source">Начало связи.</param>
/// <param name="target">Конец связи.</param>
/// <returns>Значение, определяющее существует ли связь.</returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool Exists<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
↪ source, TLinkAddress target) =>
↪ Comparer<TLinkAddress>.Default.Compare(links.Count(links.Constants.Any, source,
↪ target), default) > 0;

#region Ensure
// TODO: May be move to EnsureExtensions or make it both there and here

/// <summary>
/// <para>
/// Ensures the link exists using the specified links.
/// </para>
/// <para></para>
/// </summary>
/// <typeparam name="TLinkAddress">
/// <para>The link.</para>
/// <para></para>
/// </typeparam>
/// <param name="links">
/// <para>The links.</para>
/// <para></para>
/// </param>
/// <param name="restriction">
/// <para>The restriction.</para>
/// <para></para>
/// </param>
/// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
/// <para>sequence[{i}]</para>
/// <para></para>
/// </exception>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static void EnsureLinkExists<TLinkAddress>(this ILinks<TLinkAddress> links,
↪ IList<TLinkAddress>? restriction)
{
    for (var i = 0; i < restriction.Count; i++)
    {
        if (!links.Exists(restriction[i]))
        {
            throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(restriction[i],
↪ $"sequence[{i}]");
        }
    }
}

/// <summary>
/// <para>
/// Ensures the inner reference exists using the specified links.
/// </para>
/// <para></para>
/// </summary>
/// <typeparam name="TLinkAddress">
/// <para>The link.</para>
/// <para></para>
/// </typeparam>
/// <param name="links">
/// <para>The links.</para>
/// <para></para>
/// </param>
/// <param name="reference">
/// <para>The reference.</para>
/// <para></para>
/// </param>
/// <param name="argumentName">
/// <para>The argument name.</para>
/// <para></para>
/// </param>
/// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
/// <para></para>
/// <para></para>

```



```

570     /// </exception>
571     [MethodImpl(MethodImplOptions.AggressiveInlining)]
572     public static void EnsureInnerReferenceExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, TLinkAddress reference, string argumentName)
573     {
574         if (links.Constants.IsInternalReference(reference) && !links.Exists(reference))
575         {
576             throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(reference,
    ↪ argumentName);
577         }
578     }
579
580     /// <summary>
581     /// <para>
582     /// Ensures the inner reference exists using the specified links.
583     /// </para>
584     /// <para></para>
585     /// </summary>
586     /// <typeparam name="TLinkAddress">
587     /// <para>The link.</para>
588     /// <para></para>
589     /// </typeparam>
590     /// <param name="links">
591     /// <para>The links.</para>
592     /// <para></para>
593     /// </param>
594     /// <param name="restriction">
595     /// <para>The restriction.</para>
596     /// <para></para>
597     /// </param>
598     /// <param name="argumentName">
599     /// <para>The argument name.</para>
600     /// <para></para>
601     /// </param>
602     [MethodImpl(MethodImplOptions.AggressiveInlining)]
603     public static void EnsureInnerReferenceExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, IList<TLinkAddress>? restriction, string argumentName)
604     {
605         for (int i = 0; i < restriction.Count; i++)
606         {
607             links.EnsureInnerReferenceExists(restriction[i], argumentName);
608         }
609     }
610
611     /// <summary>
612     /// <para>
613     /// Ensures the link is any or exists using the specified links.
614     /// </para>
615     /// <para></para>
616     /// </summary>
617     /// <typeparam name="TLinkAddress">
618     /// <para>The link.</para>
619     /// <para></para>
620     /// </typeparam>
621     /// <param name="links">
622     /// <para>The links.</para>
623     /// <para></para>
624     /// </param>
625     /// <param name="restriction">
626     /// <para>The restriction.</para>
627     /// <para></para>
628     /// </param>
629     /// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
630     /// <para>sequence[{i}]</para>
631     /// <para></para>
632     /// </exception>
633     [MethodImpl(MethodImplOptions.AggressiveInlining)]
634     public static void EnsureLinkIsAnyOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, IList<TLinkAddress>? restriction)
635     {
636         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
637         var any = links.Constants.Any;
638         for (var i = 0; i < restriction.Count; i++)
639         {
640             if (!equalityComparer.Equals(restriction[i], any) &&
    ↪ !links.Exists(restriction[i]))
641             {

```

```

642         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(restriction[i],
        ↪     $"sequence[{i}]");
643     }
644 }
645 }
646
647 /// <summary>
648 /// <para>
649 /// Ensures the link is any or exists using the specified links.
650 /// </para>
651 /// <para></para>
652 /// </summary>
653 /// <typeparam name="TLinkAddress">
654 /// <para>The link.</para>
655 /// <para></para>
656 /// </typeparam>
657 /// <param name="links">
658 /// <para>The links.</para>
659 /// <para></para>
660 /// </param>
661 /// <param name="link">
662 /// <para>The link.</para>
663 /// <para></para>
664 /// </param>
665 /// <param name="argumentName">
666 /// <para>The argument name.</para>
667 /// <para></para>
668 /// </param>
669 /// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
670 /// <para></para>
671 /// <para></para>
672 /// </exception>
673 [MethodImpl(MethodImplOptions.AggressiveInlining)]
674 public static void EnsureLinkIsAnyOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, TLinkAddress link, string argumentName)
675 {
676     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
677     if (!equalityComparer.Equals(link, links.Constants.Any) && !links.Exists(link))
678     {
679         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
680     }
681 }
682
683 /// <summary>
684 /// <para>
685 /// Ensures the link is itself or exists using the specified links.
686 /// </para>
687 /// <para></para>
688 /// </summary>
689 /// <typeparam name="TLinkAddress">
690 /// <para>The link.</para>
691 /// <para></para>
692 /// </typeparam>
693 /// <param name="links">
694 /// <para>The links.</para>
695 /// <para></para>
696 /// </param>
697 /// <param name="link">
698 /// <para>The link.</para>
699 /// <para></para>
700 /// </param>
701 /// <param name="argumentName">
702 /// <para>The argument name.</para>
703 /// <para></para>
704 /// </param>
705 /// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
706 /// <para></para>
707 /// <para></para>
708 /// </exception>
709 [MethodImpl(MethodImplOptions.AggressiveInlining)]
710 public static void EnsureLinkIsItselfOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, TLinkAddress link, string argumentName)
711 {
712     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
713     if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))
714     {
715         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
716     }

```

```

717 }
718
719 /// <param name="links">Хранилище связей.</param>
720 [MethodImpl(MethodImplOptions.AggressiveInlining)]
721 public static void EnsureDoesNotExists<TLinkAddress>(this ILinks<TLinkAddress> links,
722     ↪ TLinkAddress source, TLinkAddress target)
723 {
724     if (links.Exists(source, target))
725     {
726         throw new LinkWithSameValueAlreadyExistsException();
727     }
728 }
729
730 /// <param name="links">Хранилище связей.</param>
731 [MethodImpl(MethodImplOptions.AggressiveInlining)]
732 public static void EnsureNoUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
733     ↪ TLinkAddress link)
734 {
735     if (links.HasUsages(link))
736     {
737         throw new ArgumentLinkHasDependenciesException<TLinkAddress>(link);
738     }
739 }
740
741 /// <param name="links">Хранилище связей.</param>
742 [MethodImpl(MethodImplOptions.AggressiveInlining)]
743 public static void EnsureCreated<TLinkAddress>(this ILinks<TLinkAddress> links, params
744     ↪ TLinkAddress[] addresses) => links.EnsureCreated(links.Create, addresses);
745
746 /// <param name="links">Хранилище связей.</param>
747 [MethodImpl(MethodImplOptions.AggressiveInlining)]
748 public static void EnsurePointsCreated<TLinkAddress>(this ILinks<TLinkAddress> links,
749     ↪ params TLinkAddress[] addresses) => links.EnsureCreated(links.CreatePoint,
750     ↪ addresses);
751
752 /// <param name="links">Хранилище связей.</param>
753 [MethodImpl(MethodImplOptions.AggressiveInlining)]
754 public static void EnsureCreated<TLinkAddress>(this ILinks<TLinkAddress> links,
755     ↪ Func<TLinkAddress> creator, params TLinkAddress[] addresses)
756 {
757     var addressToUInt64Converter = CheckedConverter<TLinkAddress, ulong>.Default;
758     var uInt64ToAddressConverter = CheckedConverter<ulong, TLinkAddress>.Default;
759     var nonExistentAddresses = new HashSet<TLinkAddress>(addresses.Where(x =>
760     ↪ !links.Exists(x)));
761     if (nonExistentAddresses.Count > 0)
762     {
763         var max = nonExistentAddresses.Max();
764         max = uInt64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
765     ↪ Convert(max),
766     ↪ addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
767     ↪ imum)));
768         var createdLinks = new List<TLinkAddress>();
769         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
770         TLinkAddress createdLink = creator();
771         while (!equalityComparer.Equals(createdLink, max))
772         {
773             createdLinks.Add(createdLink);
774         }
775         for (var i = 0; i < createdLinks.Count; i++)
776         {
777             if (!nonExistentAddresses.Contains(createdLinks[i]))
778             {
779                 links.Delete(createdLinks[i]);
780             }
781         }
782     }
783 }
784
785 #endregion
786
787 /// <param name="links">Хранилище связей.</param>
788 [MethodImpl(MethodImplOptions.AggressiveInlining)]
789 public static TLinkAddress CountUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
790     ↪ TLinkAddress link)
791 {
792     var constants = links.Constants;
793     var values = links.GetLink(link);

```

```

783     TLinkAddress usagesAsSource = links.Count(new Link<TLinkAddress>(constants.Any,
784         ↪ link, constants.Any));
785     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
786     if (equalityComparer.Equals(links.GetSource(values), link))
787     {
788         usagesAsSource = Arithmetic<TLinkAddress>.Decrement(usagesAsSource);
789     }
790     TLinkAddress usagesAsTarget = links.Count(new Link<TLinkAddress>(constants.Any,
791         ↪ constants.Any, link));
792     if (equalityComparer.Equals(links.GetTarget(values), link))
793     {
794         usagesAsTarget = Arithmetic<TLinkAddress>.Decrement(usagesAsTarget);
795     }
796     return Arithmetic<TLinkAddress>.Add(usagesAsSource, usagesAsTarget);
797 }
798
799 /// <param name="links">Хранилище связей.</param>
800 [MethodImpl(MethodImplOptions.AggressiveInlining)]
801 public static bool HasUsages<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
802     ↪ link) => Comparer<TLinkAddress>.Default.Compare(links.CountUsages(link), default) >
803     ↪ 0;
804
805 /// <param name="links">Хранилище связей.</param>
806 [MethodImpl(MethodImplOptions.AggressiveInlining)]
807 public static bool Equals<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
808     ↪ link, TLinkAddress source, TLinkAddress target)
809 {
810     var constants = links.Constants;
811     var values = links.GetLink(link);
812     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
813     return equalityComparer.Equals(links.GetSource(values), source) &&
814         ↪ equalityComparer.Equals(links.GetTarget(values), target);
815 }
816
817 /// <summary>
818 /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
819 /// </summary>
820 /// <param name="links">Хранилище связей.</param>
821 /// <param name="source">Индекс связи, которая является началом для искомой
822     ↪ связи.</param>
823 /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
824 /// <returns>Индекс искомой связи с указанными Source (началом) и Target
825     ↪ (концом).</returns>
826 [MethodImpl(MethodImplOptions.AggressiveInlining)]
827 public static TLinkAddress SearchOrDefault<TLinkAddress>(this ILinks<TLinkAddress>
828     ↪ links, TLinkAddress source, TLinkAddress target)
829 {
830     var constants = links.Constants;
831     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
832         ↪ constants.Break, default);
833     links.Each(setter.SetFirstAndReturnFalse, constants.Any, source, target);
834     return setter.Result;
835 }
836
837 public static TLinkAddress CreatePoint<TLinkAddress>(this ILinks<TLinkAddress> links)
838 {
839     var constants = links.Constants;
840     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
841         ↪ constants.Break);
842     links.CreatePoint(setter.SetFirstFromSecondListAndReturnTrue);
843     return setter.Result;
844 }
845
846 /// <param name="links">Хранилище связей.</param>
847 [MethodImpl(MethodImplOptions.AggressiveInlining)]
848 public static TLinkAddress CreatePoint<TLinkAddress>(this ILinks<TLinkAddress> links,
849     ↪ WriteHandler<TLinkAddress>? handler)
850 {
851     var constants = links.Constants;
852     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
853         ↪ constants.Break, handler);
854     TLinkAddress link = default;
855     TLinkAddress HandlerWrapper(ICollection<TLinkAddress>? before, ICollection<TLinkAddress>? after)
856     {
857         link = links.GetIndex(after);
858         return handlerState.Handle(before, after);
859     }
860     handlerState.Apply(links.Create(null, HandlerWrapper));
861 }

```

```

848     handlerState.Apply(links.Update(link, link, link, HandlerWrapper));
849     return handlerState.Result;
850 }
851
852 public static TLinkAddress CreateAndUpdate<TLinkAddress>(this ILinks<TLinkAddress>
853 ↪ links, TLinkAddress source, TLinkAddress target)
854 {
855     var constants = links.Constants;
856     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
857 ↪ constants.Break);
858     links.CreateAndUpdate(source, target, setter.SetFirstFromSecondListAndReturnTrue);
859     return setter.Result;
860 }
861
862 /// <param name="links">Хранилище связей.</param>
863 [MethodImpl(MethodImplOptions.AggressiveInlining)]
864 public static TLinkAddress CreateAndUpdate<TLinkAddress>(this ILinks<TLinkAddress>
865 ↪ links, TLinkAddress source, TLinkAddress target, WriteHandler<TLinkAddress>? handler)
866 {
867     var constants = links.Constants;
868     TLinkAddress createdLink = default;
869     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
870 ↪ constants.Break, handler);
871     handlerState.Apply(links.Create(null, (before, after) =>
872     {
873         createdLink = links.GetIndex(after);
874         return handlerState.Handle(before, after);
875     }));
876     handlerState.Apply(links.Update(createdLink, source, target, handler));
877     return handlerState.Result;
878 }
879
880 /// <summary>
881 /// Обновляет связь с указанными началом (Source) и концом (Target)
882 /// на связь с указанными началом (NewSource) и концом (NewTarget).
883 /// </summary>
884 /// <param name="links">Хранилище связей.</param>
885 /// <param name="link">Индекс обновляемой связи.</param>
886 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
887 ↪ выполняется обновление.</param>
888 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
889 ↪ выполняется обновление.</param>
890 /// <returns>Индекс обновлённой связи.</returns>
891 [MethodImpl(MethodImplOptions.AggressiveInlining)]
892 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
893 ↪ TLinkAddress link, TLinkAddress newSource, TLinkAddress newTarget) =>
894 ↪ links.Update(new LinkAddress<TLinkAddress>(link), new Link<TLinkAddress>(link,
895 ↪ newSource, newTarget));
896
897 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links, params
898 ↪ TLinkAddress[] restriction) => links.Update((IList<TLinkAddress>)restriction);
899
900 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
901 ↪ WriteHandler<TLinkAddress>? handler, params TLinkAddress[] restriction) =>
902 ↪ links.Update(restriction, handler);
903
904 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
905 ↪ IList<TLinkAddress>? restriction)
906 {
907     var constants = links.Constants;
908     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
909 ↪ constants.Break);
910     links.Update(restriction, setter.SetFirstFromSecondListAndReturnTrue);
911     return setter.Result;
912 }
913
914 /// <summary>
915 /// Обновляет связь с указанными началом (Source) и концом (Target)
916 /// на связь с указанными началом (NewSource) и концом (NewTarget).
917 /// </summary>
918 /// <param name="links">Хранилище связей.</param>
919 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
920 ↪ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Itself -
921 ↪ требование установить ссылку на себя, 1..∞ конкретный адрес другой связи.</param>
922 /// <returns>Индекс обновлённой связи.</returns>
923 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

910 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
911     ↳ IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>? handler)
912 {
913     return restriction.Count switch
914     {
915         2 => links.MergeAndDelete(restriction[0], restriction[1], handler),
916         4 => links.UpdateOrCreateOrGet(restriction[0], restriction[1], restriction[2],
917             ↳ restriction[3], handler),
918         _ => links.Update(restriction[0], restriction[1], restriction[2], handler)
919     };
920 }

921 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
922     TLinkAddress link, TLinkAddress newSource, TLinkAddress newTarget,
923     ↳ WriteHandler<TLinkAddress>? handler) => links.Update(new
924     ↳ LinkAddress<TLinkAddress>(link), new Link<TLinkAddress>(link, newSource, newTarget),
925     ↳ handler);

926 /// <summary>
927 /// <para>
928 /// Resolves the constant as self reference using the specified links.
929 /// </para>
930 /// <para></para>
931 /// </summary>
932 /// <typeparam name="TLinkAddress">
933 /// <para>The link.</para>
934 /// <para></para>
935 /// </typeparam>
936 /// <param name="links">
937 /// <para>The links.</para>
938 /// <para></para>
939 /// </param>
940 /// <param name="constant">
941 /// <para>The constant.</para>
942 /// <para></para>
943 /// </param>
944 /// <param name="substitution">
945 /// <para>The substitution.</para>
946 /// <para></para>
947 /// </param>
948 /// <returns>
949 /// <para>A list of t link</para>
950 /// <para></para>
951 /// </returns>
952 [MethodImpl(MethodImplOptions.AggressiveInlining)]
953 public static IList<TLinkAddress>? ResolveConstantAsSelfReference<TLinkAddress>(this
954     ↳ ILinks<TLinkAddress> links, TLinkAddress constant, IList<TLinkAddress>? restriction,
955     ↳ IList<TLinkAddress>? substitution)
956 {
957     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
958     var constants = links.Constants;
959     var restrictionIndex = links.GetIndex(restriction);
960     var substitutionIndex = links.GetIndex(substitution);
961     if (equalityComparer.Equals(substitutionIndex, default))
962     {
963         substitutionIndex = restrictionIndex;
964     }
965     var source = links.GetSource(substitution);
966     var target = links.GetTarget(substitution);
967     source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;
968     target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
969     return new Link<TLinkAddress>(substitutionIndex, source, target);
970 }

971 /// <summary>
972 /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
973   ↳ с указанными Source (началом) и Target (концом).
974 /// </summary>
975 /// <param name="links">Хранилище связей.</param>
976 /// <param name="source">Индекс связи, которая является началом на создаваемой
977   ↳ связи.</param>
978 /// <param name="target">Индекс связи, которая является концом для создаваемой
979   ↳ связи.</param>
980 /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>

```

```

977 [MethodImpl(MethodImplOptions.AggressiveInlining)]
978 public static TLinkAddress GetOrCreate<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress source, TLinkAddress target)
979 {
980     var link = links.SearchOrDefault(source, target);
981     if (EqualityComparer<TLinkAddress>.Default.Equals(link, default))
982     {
983         link = links.CreateAndUpdate(source, target);
984     }
985     return link;
986 }
987
988 public static TLinkAddress UpdateOrCreateOrGet<TLinkAddress>(this ILinks<TLinkAddress>
    ↳ links, TLinkAddress source, TLinkAddress target, TLinkAddress newSource,
    ↳ TLinkAddress newTarget)
989 {
990     var constants = links.Constants;
991     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
    ↳ constants.Break);
992     links.UpdateOrCreateOrGet(source, target, newSource, newTarget,
    ↳ setter.SetFirstFromSecondListAndReturnTrue);
993     return setter.Result;
994 }
995
996 /// <summary>
997 /// Обновляет связь с указанными началом (Source) и концом (Target)
998 /// на связь с указанными началом (NewSource) и концом (NewTarget).
999 /// </summary>
1000 /// <param name="links">Хранилище связей.</param>
1001 /// <param name="source">Индекс связи, которая является началом обновляемой
    ↳ связи.</param>
1002 /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
1003 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
    ↳ выполняется обновление.</param>
1004 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
    ↳ выполняется обновление.</param>
1005 /// <returns>Индекс обновлённой связи.</returns>
1006 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1007 public static TLinkAddress UpdateOrCreateOrGet<TLinkAddress>(this ILinks<TLinkAddress>
    ↳ links, TLinkAddress source, TLinkAddress target, TLinkAddress newSource,
    ↳ TLinkAddress newTarget, WriteHandler<TLinkAddress>? handler)
1008 {
1009     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1010     var link = links.SearchOrDefault(source, target);
1011     if (equalityComparer.Equals(link, default))
1012     {
1013         return links.CreateAndUpdate(newSource, newTarget, handler);
1014     }
1015     if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
    ↳ target))
1016     {
1017         var linkStruct = new Link<TLinkAddress>(link, source, target);
1018         return link;
1019     }
1020     return links.Update(link, newSource, newTarget, handler);
1021 }
1022
1023 /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1024 /// <param name="links">Хранилище связей.</param>
1025 /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
1026 /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
1027 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1028 public static TLinkAddress DeleteIfExists<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ TLinkAddress source, TLinkAddress target)
1029 {
1030     var link = links.SearchOrDefault(source, target);
1031     if (!EqualityComparer<TLinkAddress>.Default.Equals(link, default))
1032     {
1033         links.Delete(link);
1034         return link;
1035     }
1036     return default;
1037 }
1038
1039 /// <summary>Удаляет несколько связей.</summary>
1040 /// <param name="links">Хранилище связей.</param>
1041 /// <param name="deletedLinks">Список адресов связей к удалению.</param>
1042 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

1043 public static void DeleteMany<TLinkAddress>(this ILinks<TLinkAddress> links,
1044     ↳ IList<TLinkAddress>? deletedLinks)
1045 {
1046     for (int i = 0; i < deletedLinks.Count; i++)
1047     {
1048         links.Delete(deletedLinks[i]);
1049     }
1050 }
1051 public static void DeleteAllUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
1052     ↳ TLinkAddress linkIndex) => links.DeleteAllUsages(linkIndex, null);
1053
1054 /// <remarks>Before execution of this method ensure that deleted link is detached (all
1055     ↳ values - source and target are reset to null) or it might enter into infinite
1056     ↳ recursion.</remarks>
1057 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1058 public static TLinkAddress DeleteAllUsages<TLinkAddress>(this ILinks<TLinkAddress>
1059     ↳ links, TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1060 {
1061     var constants = links.Constants;
1062     var any = constants.Any;
1063     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1064     var usagesAsSourceQuery = new Link<TLinkAddress>(any, linkIndex, any);
1065     var usagesAsTargetQuery = new Link<TLinkAddress>(any, any, linkIndex);
1066     var usages = new List<IList<TLinkAddress>?>();
1067     var usagesFiller = new ListFiller<IList<TLinkAddress>?, TLinkAddress>(usages,
1068     ↳ constants.Continue);
1069     links.Each(usagesFiller.AddAndReturnConstant, usagesAsSourceQuery);
1070     links.Each(usagesFiller.AddAndReturnConstant, usagesAsTargetQuery);
1071     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
1072     ↳ constants.Break, handler);
1073     foreach (var usage in usages)
1074     {
1075         if (equalityComparer.Equals(links.GetIndex(usage), linkIndex) ||
1076             ↳ !links.Exists(links.GetIndex(usage)))
1077         {
1078             continue;
1079         }
1080         handlerState.Apply(links.Delete(links.GetIndex(usage), handlerState.Handler));
1081     }
1082     return handlerState.Result;
1083 }
1084
1085 /// <summary>
1086 /// <para>
1087 /// Deletes the by query using the specified links.
1088 /// </para>
1089 /// <para></para>
1090 /// </summary>
1091 /// <typeparam name="TLinkAddress">
1092 /// <para>The link.</para>
1093 /// <para></para>
1094 /// </typeparam>
1095 /// <param name="links">
1096 /// <para>The links.</para>
1097 /// <para></para>
1098 /// </param>
1099 /// <param name="query">
1100 /// <para>The query.</para>
1101 /// <para></para>
1102 /// </param>
1103 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1104 public static void DeleteByQuery<TLinkAddress>(this ILinks<TLinkAddress> links,
1105     ↳ Link<TLinkAddress> query)
1106 {
1107     var queryResult = new List<TLinkAddress>();
1108     var queryResultFiller = new ListFiller<TLinkAddress, TLinkAddress>(queryResult,
1109     ↳ links.Constants.Continue);
1110     links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
1111     foreach (var link in queryResult)
1112     {
1113         links.Delete(link);
1114     }
1115 }
1116
1117 // TODO: Move to Platform.Data
1118 /// <summary>
1119

```



```

1110     /// <para>
1111     /// Determines whether are values reset.
1112     /// </para>
1113     /// <para></para>
1114     /// </summary>
1115     /// <typeparam name="TLinkAddress">
1116     /// <para>The link.</para>
1117     /// <para></para>
1118     /// </typeparam>
1119     /// <param name="links">
1120     /// <para>The links.</para>
1121     /// <para></para>
1122     /// </param>
1123     /// <param name="linkIndex">
1124     /// <para>The link index.</para>
1125     /// <para></para>
1126     /// </param>
1127     /// <returns>
1128     /// <para>The bool</para>
1129     /// <para></para>
1130     /// </returns>
1131     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1132     public static bool AreValuesReset<TLinkAddress>(this ILinks<TLinkAddress> links,
1133     ↪ TLinkAddress linkIndex)
1134     {
1135         var nullConstant = links.Constants.Null;
1136         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1137         var link = links.GetLink(linkIndex);
1138         for (int i = 1; i < link.Count; i++)
1139         {
1140             if (!equalityComparer.Equals(link[i], nullConstant))
1141             {
1142                 return false;
1143             }
1144         }
1145         return true;
1146     }
1147
1148     public static void ResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1149     ↪ TLinkAddress linkIndex) => links.ResetValues(linkIndex, null);
1150
1151     // TODO: Create a universal version of this method in Platform.Data (with using of for
1152     ↪ loop)
1153     /// <summary>
1154     /// <para>
1155     /// Resets the values using the specified links.
1156     /// </para>
1157     /// <para></para>
1158     /// </summary>
1159     /// <typeparam name="TLinkAddress">
1160     /// <para>The link.</para>
1161     /// <para></para>
1162     /// </typeparam>
1163     /// <param name="links">
1164     /// <para>The links.</para>
1165     /// <para></para>
1166     /// </param>
1167     /// <param name="linkIndex">
1168     /// <para>The link index.</para>
1169     /// <para></para>
1170     /// </param>
1171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1172     public static TLinkAddress ResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1173     ↪ TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1174     {
1175         var nullConstant = links.Constants.Null;
1176         var updateRequest = new Link<TLinkAddress>(linkIndex, nullConstant, nullConstant);
1177         return links.Update(updateRequest, handler);
1178     }
1179
1180     public static void EnforceResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1181     ↪ TLinkAddress linkIndex) => links.EnforceResetValues(linkIndex, null);
1182
1183     // TODO: Create a universal version of this method in Platform.Data (with using of for
1184     ↪ loop)
1185     /// <summary>
1186     /// <para>

```

```

1182     /// Enforces the reset values using the specified links.
1183     /// </para>
1184     /// <para></para>
1185     /// </summary>
1186     /// <typeparam name="TLinkAddress">
1187     /// <para>The link.</para>
1188     /// <para></para>
1189     /// </typeparam>
1190     /// <param name="links">
1191     /// <para>The links.</para>
1192     /// <para></para>
1193     /// </param>
1194     /// <param name="linkIndex">
1195     /// <para>The link index.</para>
1196     /// <para></para>
1197     /// </param>
1198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1199     public static TLinkAddress EnforceResetValues<TLinkAddress>(this ILinks<TLinkAddress>
1200     ↪ links, TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1201     {
1202         if (!links.AreValuesReset(linkIndex))
1203         {
1204             return links.ResetValues(linkIndex, handler);
1205         }
1206         return links.Constants.Continue;
1207     }
1208
1209     public static void MergeUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
1210     ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex) =>
1211     ↪ links.MergeUsages(oldLinkIndex, newLinkIndex, null);
1212
1213     /// <summary>
1214     /// Merging two usages graphs, all children of old link moved to be children of new link
1215     ↪ or deleted.
1216     /// </summary>
1217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1218     public static TLinkAddress MergeUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
1219     ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex, WriteHandler<TLinkAddress>?
1220     ↪ handler)
1221     {
1222         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1223         if (equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1224         {
1225             return newLinkIndex;
1226         }
1227         var constants = links.Constants;
1228         var usagesAsSource = links.All(new Link<TLinkAddress>(constants.Any, oldLinkIndex,
1229         ↪ constants.Any));
1230         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
1231         ↪ constants.Break, handler);
1232         for (var i = 0; i < usagesAsSource.Count; i++)
1233         {
1234             var usageAsSource = usagesAsSource[i];
1235             if (equalityComparer.Equals(links.GetIndex(usageAsSource), oldLinkIndex))
1236             {
1237                 continue;
1238             }
1239             var restriction = new LinkAddress<TLinkAddress>(links.GetIndex(usageAsSource));
1240             var substitution = new Link<TLinkAddress>(newLinkIndex,
1241             ↪ links.GetTarget(usageAsSource));
1242             handlerState.Apply(links.Update(restriction, substitution,
1243             ↪ handlerState.Handler));
1244         }
1245         var usagesAsTarget = links.All(new Link<TLinkAddress>(constants.Any, constants.Any,
1246         ↪ oldLinkIndex));
1247         for (var i = 0; i < usagesAsTarget.Count; i++)
1248         {
1249             var usageAsTarget = usagesAsTarget[i];
1250             if (equalityComparer.Equals(links.GetIndex(usageAsTarget), oldLinkIndex))
1251             {
1252                 continue;
1253             }
1254             var restriction = links.GetLink(links.GetIndex(usageAsTarget));
1255             var substitution = new Link<TLinkAddress>(links.GetTarget(usageAsTarget),
1256             ↪ newLinkIndex);
1257             handlerState.Apply(links.Update(restriction, substitution,
1258             ↪ handlerState.Handler));
1259         }
1260     }

```

```

1247         return handlerState.Result;
1248     }
1249
1250     public static TLinkAddress MergeAndDelete<TLinkAddress>(this ILinks<TLinkAddress> links,
1251     ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex)
1252     {
1253         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1254         if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1255         {
1256             links.MergeUsages(oldLinkIndex, newLinkIndex);
1257             links.Delete(oldLinkIndex);
1258         }
1259         return newLinkIndex;
1260     }
1261
1262     /// <summary>
1263     /// Replace one link with another (replaced link is deleted, children are updated or
1264     ↪ deleted).
1265     /// </summary>
1266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1267     public static TLinkAddress MergeAndDelete<TLinkAddress>(this ILinks<TLinkAddress> links,
1268     ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex, WriteHandler<TLinkAddress>?
1269     ↪ handler)
1270     {
1271         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1272         var constants = links.Constants;
1273         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
1274     ↪ constants.Break, handler);
1275         if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1276         {
1277             handlerState.Apply(links.MergeUsages(oldLinkIndex, newLinkIndex,
1278     ↪ handlerState.Handler));
1279             handlerState.Apply(links.Delete(oldLinkIndex, handlerState.Handler));
1280         }
1281         return handlerState.Result;
1282     }
1283
1284     /// <summary>
1285     /// <para>
1286     /// Decorates the with automatic uniqueness and usages resolution using the specified
1287     ↪ links.
1288     /// </para>
1289     /// <para></para>
1290     /// </summary>
1291     /// <typeparam name="TLinkAddress">
1292     /// <para>The link.</para>
1293     /// <para></para>
1294     /// </typeparam>
1295     /// <param name="links">
1296     /// <para>The links.</para>
1297     /// <para></para>
1298     /// </param>
1299     /// <returns>
1300     /// <para>The links.</para>
1301     /// <para></para>
1302     /// </returns>
1303     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1304     public static ILinks<TLinkAddress>
1305     ↪ DecorateWithAutomaticUniquenessAndUsagesResolution<TLinkAddress>(this
1306     ↪ ILinks<TLinkAddress> links)
1307     {
1308         links = new LinksCascadeUsagesResolver<TLinkAddress>(links);
1309         links = new NonNullContentsLinkDeletionResolver<TLinkAddress>(links);
1310         links = new LinksCascadeUniquenessAndUsagesResolver<TLinkAddress>(links);
1311         return links;
1312     }
1313
1314     /// <summary>
1315     /// <para>
1316     /// Formats the links.
1317     /// </para>
1318     /// <para></para>
1319     /// </summary>
1320     /// <typeparam name="TLinkAddress">
1321     /// <para>The link.</para>
1322     /// <para></para>
1323     /// </typeparam>
1324     /// <param name="links">

```

```

1316     /// <para>The links.</para>
1317     /// <para></para>
1318     /// </param>
1319     /// <param name="link">
1320     /// <para>The link.</para>
1321     /// <para></para>
1322     /// </param>
1323     /// <returns>
1324     /// <para>The string</para>
1325     /// <para></para>
1326     /// </returns>
1327     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1328     public static string Format<TLinkAddress>(this ILinks<TLinkAddress> links,
1329     ↪ IList<TLinkAddress>? link)
1330     {
1331         var constants = links.Constants;
1332         return $"({links.GetIndex(link)}: {links.GetSource(link)} {links.GetTarget(link)})";
1333     }
1334     /// <summary>
1335     /// <para>
1336     /// Formats the 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     /// <param name="link">
1349     /// <para>The link.</para>
1350     /// <para></para>
1351     /// </param>
1352     /// <returns>
1353     /// <para>The string</para>
1354     /// <para></para>
1355     /// </returns>
1356     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1357     public static string Format<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
1358     ↪ link) => links.Format(links.GetLink(link));
1359 }

```

1.25 ./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}},
12     ↪ LinksConstants{TLinkAddress}" />
13     /// <seealso cref="ILinks{TLinkAddress}" />
14     public interface ISynchronizedLinks<TLinkAddress> : ISynchronizedLinks<TLinkAddress,
15     ↪ ILinks<TLinkAddress>, LinksConstants<TLinkAddress>>, ILinks<TLinkAddress>
16     {
17     }
18 }

```

1.26 ./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>>,
18         ↳ IReadOnlyList<TLinkAddress>, IList<TLinkAddress>
19     {
20         /// <summary>
21         /// <para>
22         /// The link.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         public static readonly Link<TLinkAddress> Null = new Link<TLinkAddress>();
27         private static readonly LinksConstants<TLinkAddress> _constants =
28             ↳ Default<LinksConstants<TLinkAddress>>.Instance;
29         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
30             ↳ EqualityComparer<TLinkAddress>.Default;
31         private const int Length = 3;
32
33         /// <summary>
34         /// <para>
35         /// The index.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         public readonly TLinkAddress Index;
40         /// <summary>
41         /// <para>
42         /// The source.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         public readonly TLinkAddress Source;
47         /// <summary>
48         /// <para>
49         /// The target.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         public readonly TLinkAddress Target;
54
55         /// <summary>
56         /// <para>
57         /// Initializes a new <see cref="Link"/> instance.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         /// <param name="values">
62         /// <para>A values.</para>
63         /// <para></para>
64         /// </param>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public Link(params TLinkAddress[] values) => SetValues(values, out Index, out Source,
67             ↳ out Target);
68
69         /// <summary>
70         /// <para>
71         /// Initializes a new <see cref="Link"/> instance.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         /// <param name="values">
76         /// <para>A values.</para>
77         /// <para></para>
78         /// </param>
79         [MethodImpl(MethodImplOptions.AggressiveInlining)]
80         public Link(IList<TLinkAddress>? values) => SetValues(values, out Index, out Source, out
81             ↳ Target);
82
83         /// <summary>
84         /// <para>
85         /// Initializes a new <see cref="Link"/> instance.
86         /// </para>
87         /// <para></para>
88         /// </summary>

```

```

84    /// <param name="other">
85    /// <para>A other.</para>
86    /// </para>
87    /// </param>
88    /// <exception cref="NotSupportedException">
89    /// <para></para>
90    /// </para>
91    /// </exception>
92    [MethodImpl(MethodImplOptions.AggressiveInlining)]
93    public Link(object other)
94    {
95        if (other is Link<TLinkAddress> otherLink)
96        {
97            SetValues(ref otherLink, out Index, out Source, out Target);
98        }
99        else if (other is IList<TLinkAddress> otherList)
100        {
101            SetValues(otherList, out Index, out Source, out Target);
102        }
103        else
104        {
105            throw new NotSupportedException();
106        }
107    }
108
109    /// <summary>
110    /// <para>
111    /// Initializes a new <see cref="Link"/> instance.
112    /// </para>
113    /// </summary>
114    /// <param name="other">
115    /// <para>A other.</para>
116    /// </para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    public Link(ref Link<TLinkAddress> other) => SetValues(ref other, out Index, out Source,
120    ↪ out Target);
121
122    /// <summary>
123    /// <para>
124    /// Initializes a new <see cref="Link"/> instance.
125    /// </para>
126    /// </summary>
127    /// <param name="index">
128    /// <para>A index.</para>
129    /// </para>
130    /// </param>
131    /// <param name="source">
132    /// <para>A source.</para>
133    /// </para>
134    /// </param>
135    /// <param name="target">
136    /// <para>A target.</para>
137    /// </para>
138    /// </param>
139    [MethodImpl(MethodImplOptions.AggressiveInlining)]
140    public Link(TLinkAddress index, TLinkAddress source, TLinkAddress target)
141    {
142        Index = index;
143        Source = source;
144        Target = target;
145    }
146
147    [MethodImpl(MethodImplOptions.AggressiveInlining)]
148    private static void SetValues(ref Link<TLinkAddress> other, out TLinkAddress index, out
149    ↪ TLinkAddress source, out TLinkAddress target)
150    {
151        index = other.Index;
152        source = other.Source;
153        target = other.Target;
154    }
155
156    [MethodImpl(MethodImplOptions.AggressiveInlining)]
157    private static void SetValues(IList<TLinkAddress>? values, out TLinkAddress index, out
158    ↪ TLinkAddress source, out TLinkAddress target)
159    {
160        if (values == null)
161        {

```

```

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,
250         ↪ other.Index)
251         && _equalityComparer.Equals(Source, other.Source)
252         && _equalityComparer.Equals(Target, other.Target);
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
277         ↪ target) => $"{({index}: {source}->{target})}";
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     [MethodImpl(MethodImplOptions.AggressiveInlining)]
300     public static implicit operator TLinkAddress[] (Link<TLinkAddress> link) =>
301         ↪ link.ToArray();
302     [MethodImpl(MethodImplOptions.AggressiveInlining)]
303     public static implicit operator Link<TLinkAddress> (TLinkAddress[] linkArray) => new
304         ↪ Link<TLinkAddress>(linkArray);
305     /// <summary>
306     /// <para>
307     /// Returns the string.
308     /// </para>

```



```

309     /// <para></para>
310     /// </summary>
311     /// <returns>
312     /// <para>The string</para>
313     /// <para></para>
314     /// </returns>
315     [MethodImpl(MethodImplOptions.AggressiveInlining)]
316     public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
        ↳ ToString(Source, Target) : ToString(Index, Source, Target);
317
318     #region IList
319
320     /// <summary>
321     /// <para>
322     /// Gets the count value.
323     /// </para>
324     /// <para></para>
325     /// </summary>
326     public int Count
327     {
328         [MethodImpl(MethodImplOptions.AggressiveInlining)]
329         get => Length;
330     }
331
332     /// <summary>
333     /// <para>
334     /// Gets the is read only value.
335     /// </para>
336     /// <para></para>
337     /// </summary>
338     public bool IsReadOnly
339     {
340         [MethodImpl(MethodImplOptions.AggressiveInlining)]
341         get => true;
342     }
343
344     /// <summary>
345     /// <para>
346     /// The not supported exception.
347     /// </para>
348     /// <para></para>
349     /// </summary>
350     public TLinkAddress this[int index]
351     {
352         [MethodImpl(MethodImplOptions.AggressiveInlining)]
353         get
354         {
355             Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
        ↳ nameof(index));
356             if (index == _constants.IndexPart)
357             {
358                 return Index;
359             }
360             if (index == _constants.SourcePart)
361             {
362                 return Source;
363             }
364             if (index == _constants.TargetPart)
365             {
366                 return Target;
367             }
368             throw new NotSupportedException(); // Impossible path due to
        ↳ Ensure.ArgumentInRange
369         }
370         [MethodImpl(MethodImplOptions.AggressiveInlining)]
371         set => throw new NotSupportedException();
372     }
373
374     /// <summary>
375     /// <para>
376     /// Gets the enumerator.
377     /// </para>
378     /// <para></para>
379     /// </summary>
380     /// <returns>
381     /// <para>The enumerator</para>
382     /// <para></para>
383     /// </returns>
384     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

385     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
386
387     /// <summary>
388     /// <para>
389     /// Gets the enumerator.
390     /// </para>
391     /// <para></para>
392     /// </summary>
393     /// <returns>
394     /// <para>An enumerator of t link</para>
395     /// <para></para>
396     /// </returns>
397     [MethodImpl(MethodImplOptions.AggressiveInlining)]
398     public IEnumerator<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),
        ↳ nameof(arrayIndex));
467     if (arrayIndex + Length > array.Length)
468     {
469         throw new InvalidOperationException();
470     }
471     array[arrayIndex++] = Index;
472     array[arrayIndex++] = Source;
473     array[arrayIndex] = Target;
474 }
475
476 /// <summary>
477 /// <para>
478 /// Determines whether this instance remove.
479 /// </para>
480 /// <para></para>
481 /// </summary>
482 /// <param name="item">
483 /// <para>The item.</para>
484 /// <para></para>
485 /// </param>
486 /// <returns>
487 /// <para>The bool</para>
488 /// <para></para>
489 /// </returns>
490 [MethodImpl(MethodImplOptions.AggressiveInlining)]
491 public bool Remove(TLinkAddress item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
492
493 /// <summary>
494 /// <para>
495 /// Indexes the of using the specified item.
496 /// </para>
497 /// <para></para>
498 /// </summary>
499 /// <param name="item">
500 /// <para>The item.</para>
501 /// <para></para>
502 /// </param>
503 /// <returns>
504 /// <para>The int</para>
505 /// <para></para>
506 /// </returns>
507 [MethodImpl(MethodImplOptions.AggressiveInlining)]
508 public int IndexOf(TLinkAddress item)
509 {
510     if (_equalityComparer.Equals(Index, item))
511     {
512         return _constants.IndexPart;
513     }
514     if (_equalityComparer.Equals(Source, item))
515     {
516         return _constants.SourcePart;
517     }
518     if (_equalityComparer.Equals(Target, item))
519     {
520         return _constants.TargetPart;
521     }
522     return -1;
523 }
524
525 /// <summary>
526 /// <para>
527 /// Inserts the index.
528 /// </para>
529 /// <para></para>
530 /// </summary>
531 /// <param name="index">
532 /// <para>The index.</para>
533 /// <para></para>
534 /// </param>
535 /// <param name="item">
536 /// <para>The item.</para>
537 /// <para></para>
538 /// </param>
539 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

540     public void Insert(int index, 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 }

```

1.27 ./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>

```

```

50     /// <returns>
51     /// <para>The bool</para>
52     /// <para></para>
53     /// </returns>
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     public static bool IsPartialPoint<TLinkAddress>(this Link<TLinkAddress> link) =>
        ↪ Point<TLinkAddress>.IsPartialPoint(link);
56 }
57 }

```

1.28 ./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 }

```

1.29 ./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 }

```

1.30 ./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 }

```

1.31 ./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 }

```

1.32 ./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>
31         /// The allocated links.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLinkAddress AllocatedLinks;
36
37         /// <summary>
38         /// <para>
39         /// The reserved links.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLinkAddress ReservedLinks;
44
45         /// <summary>
46         /// <para>
47         /// The free links.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public TLinkAddress FreeLinks;
52
53         /// <summary>
54         /// <para>
55         /// The first free link.
56         /// </para>
57         /// </summary>
58         public TLinkAddress FirstFreeLink;
59
60         /// <summary>
61         /// <para>
62         /// The last free link.
63         /// </para>
64         /// </summary>
65         public TLinkAddress LastFreeLink;
66
67         /// <summary>
68         /// <para>
69         /// The first allocated link.
70         /// </para>
71         /// </summary>
72         public TLinkAddress FirstAllocatedLink;
73
74         /// <summary>
75         /// <para>
76         /// The last allocated link.
77         /// </para>
78         /// </summary>
79         public TLinkAddress LastAllocatedLink;
80
81         /// <summary>
82         /// <para>
83         /// The first reserved link.
84         /// </para>
85         /// </summary>
86         public TLinkAddress FirstReservedLink;
87
88         /// <summary>
89         /// <para>
90         /// The last reserved link.
91         /// </para>
92         /// </summary>
93         public TLinkAddress LastReservedLink;
94
95         /// <summary>
96         /// <para>
97         /// The first free link.
98         /// </para>
99         /// </summary>
100        public TLinkAddress FirstFreeLink;
101
102        /// <summary>
103        /// <para>
104        /// The last free link.
105        /// </para>
106        /// </summary>
107        public TLinkAddress LastFreeLink;
108
109        /// <summary>
110        /// <para>
111        /// The first allocated link.
112        /// </para>
113        /// </summary>
114        public TLinkAddress FirstAllocatedLink;
115
116        /// <summary>
117        /// <para>
118        /// The last allocated link.
119        /// </para>
120        /// </summary>
121        public TLinkAddress LastAllocatedLink;
122
123        /// <summary>
124        /// <para>
125        /// The first reserved link.
126        /// </para>
127        /// </summary>
128        public TLinkAddress FirstReservedLink;
129
130        /// <summary>
131        /// <para>
132        /// The last reserved link.
133        /// </para>
134        /// </summary>
135        public TLinkAddress LastReservedLink;
136
137        /// <summary>
138        /// <para>
139        /// The first free link.
140        /// </para>
141        /// </summary>
142        public TLinkAddress FirstFreeLink;
143
144        /// <summary>
145        /// <para>
146        /// The last free link.
147        /// </para>
148        /// </summary>
149        public TLinkAddress LastFreeLink;
150
151        /// <summary>
152        /// <para>
153        /// The first allocated link.
154        /// </para>
155        /// </summary>
156        public TLinkAddress FirstAllocatedLink;
157
158        /// <summary>
159        /// <para>
160        /// The last allocated link.
161        /// </para>
162        /// </summary>
163        public TLinkAddress LastAllocatedLink;
164
165        /// <summary>
166        /// <para>
167        /// The first reserved link.
168        /// </para>
169        /// </summary>
170        public TLinkAddress FirstReservedLink;
171
172        /// <summary>
173        /// <para>
174        /// The last reserved link.
175        /// </para>
176        /// </summary>
177        public TLinkAddress LastReservedLink;
178
179        /// <summary>
180        /// <para>
181        /// The first free link.
182        /// </para>
183        /// </summary>
184        public TLinkAddress FirstFreeLink;
185
186        /// <summary>
187        /// <para>
188        /// The last free link.
189        /// </para>
190        /// </summary>
191        public TLinkAddress LastFreeLink;
192
193        /// <summary>
194        /// <para>
195        /// The first allocated link.
196        /// </para>
197        /// </summary>
198        public TLinkAddress FirstAllocatedLink;
199
200        /// <summary>
201        /// <para>
202        /// The last allocated link.
203        /// </para>
204        /// </summary>
205        public TLinkAddress LastAllocatedLink;
206
207        /// <summary>
208        /// <para>
209        /// The first reserved link.
210        /// </para>
211        /// </summary>
212        public TLinkAddress FirstReservedLink;
213
214        /// <summary>
215        /// <para>
216        /// The last reserved link.
217        /// </para>
218        /// </summary>
219        public TLinkAddress LastReservedLink;
220
221        /// <summary>
222        /// <para>
223        /// The first free link.
224        /// </para>
225        /// </summary>
226        public TLinkAddress FirstFreeLink;
227
228        /// <summary>
229        /// <para>
230        /// The last free link.
231        /// </para>
232        /// </summary>
233        public TLinkAddress LastFreeLink;
234
235        /// <summary>
236        /// <para>
237        /// The first allocated link.
238        /// </para>
239        /// </summary>
240        public TLinkAddress FirstAllocatedLink;
241
242        /// <summary>
243        /// <para>
244        /// The last allocated link.
245        /// </para>
246        /// </summary>
247        public TLinkAddress LastAllocatedLink;
248
249        /// <summary>
250        /// <para>
251        /// The first reserved link.
252        /// </para>
253        /// </summary>
254        public TLinkAddress FirstReservedLink;
255
256        /// <summary>
257        /// <para>
258        /// The last reserved link.
259        /// </para>
260        /// </summary>
261        public TLinkAddress LastReservedLink;
262
263        /// <summary>
264        /// <para>
265        /// The first free link.
266        /// </para>
267        /// </summary>
268        public TLinkAddress FirstFreeLink;
269
270        /// <summary>
271        /// <para>
272        /// The last free link.
273        /// </para>
274        /// </summary>
275        public TLinkAddress LastFreeLink;
276
277        /// <summary>
278        /// <para>
279        /// The first allocated link.
280        /// </para>
281        /// </summary>
282        public TLinkAddress FirstAllocatedLink;
283
284        /// <summary>
285        /// <para>
286        /// The last allocated link.
287        /// </para>
288        /// </summary>
289        public TLinkAddress LastAllocatedLink;
290
291        /// <summary>
292        /// <para>
293        /// The first reserved link.
294        /// </para>
295        /// </summary>
296        public TLinkAddress FirstReservedLink;
297
298        /// <summary>
299        /// <para>
300        /// The last reserved link.
301        /// </para>
302        /// </summary>
303        public TLinkAddress LastReservedLink;
304
305        /// <summary>
306        /// <para>
307        /// The first free link.
308        /// </para>
309        /// </summary>
310        public TLinkAddress FirstFreeLink;
311
312        /// <summary>
313        /// <para>
314        /// The last free link.
315        /// </para>
316        /// </summary>
317        public TLinkAddress LastFreeLink;
318
319        /// <summary>
320        /// <para>
321        /// The first allocated link.
322        /// </para>
323        /// </summary>
324        public TLinkAddress FirstAllocatedLink;
325
326        /// <summary>
327        /// <para>
328        /// The last allocated link.
329        /// </para>
330        /// </summary>
331        public TLinkAddress LastAllocatedLink;
332
333        /// <summary>
334        /// <para>
335        /// The first reserved link.
336        /// </para>
337        /// </summary>
338        public TLinkAddress FirstReservedLink;
339
340        /// <summary>
341        /// <para>
342        /// The last reserved link.
343        /// </para>
344        /// </summary>
345        public TLinkAddress LastReservedLink;
346
347        /// <summary>
348        /// <para>
349        /// The first free link.
350        /// </para>
351        /// </summary>
352        public TLinkAddress FirstFreeLink;
353
354        /// <summary>
355        /// <para>
356        /// The last free link.
357        /// </para>
358        /// </summary>
359        public TLinkAddress LastFreeLink;
360
361        /// <summary>
362        /// <para>
363        /// The first allocated link.
364        /// </para>
365        /// </summary>
366        public TLinkAddress FirstAllocatedLink;
367
368        /// <summary>
369        /// <para>
370        /// The last allocated link.
371        /// </para>
372        /// </summary>
373        public TLinkAddress LastAllocatedLink;
374
375        /// <summary>
376        /// <para>
377        /// The first reserved link.
378        /// </para>
379        /// </summary>
380        public TLinkAddress FirstReservedLink;
381
382        /// <summary>
383        /// <para>
384        /// The last reserved link.
385        /// </para>
386        /// </summary>
387        public TLinkAddress LastReservedLink;
388
389        /// <summary>
390        /// <para>
391        /// The first free link.
392        /// </para>
393        /// </summary>
394        public TLinkAddress FirstFreeLink;
395
396        /// <summary>
397        /// <para>
398        /// The last free link.
399        /// </para>
400        /// </summary>
401        public TLinkAddress LastFreeLink;
402
403        /// <summary>
404        /// <para>
405        /// The first allocated link.
406        /// </para>
407        /// </summary>
408        public TLinkAddress FirstAllocatedLink;
409
410        /// <summary>
411        /// <para>
412        /// The last allocated link.
413        /// </para>
414        /// </summary>
415        public TLinkAddress LastAllocatedLink;
416
417        /// <summary>
418        /// <para>
419        /// The first reserved link.
420        /// </para>
421        /// </summary>
422        public TLinkAddress FirstReservedLink;
423
424        /// <summary>
425        /// <para>
426        /// The last reserved link.
427        /// </para>
428        /// </summary>
429        public TLinkAddress LastReservedLink;
430
431        /// <summary>
432        /// <para>
433        /// The first free link.
434        /// </para>
435        /// </summary>
436        public TLinkAddress FirstFreeLink;
437
438        /// <summary>
439        /// <para>
440        /// The last free link.
441        /// </para>
442        /// </summary>
443        public TLinkAddress LastFreeLink;
444
445        /// <summary>
446        /// <para>
447        /// The first allocated link.
448        /// </para>
449        /// </summary>
450        public TLinkAddress FirstAllocatedLink;
451
452        /// <summary>
453        /// <para>
454        /// The last allocated link.
455        /// </para>
456        /// </summary>
457        public TLinkAddress LastAllocatedLink;
458
459        /// <summary>
460        /// <para>
461        /// The first reserved link.
462        /// </para>
463        /// </summary>
464        public TLinkAddress FirstReservedLink;
465
466        /// <summary>
467        /// <para>
468        /// The last reserved link.
469        /// </para>
470        /// </summary>
471        public TLinkAddress LastReservedLink;
472
473        /// <summary>
474        /// <para>
475        /// The first free link.
476        /// </para>
477        /// </summary>
478        public TLinkAddress FirstFreeLink;
479
480        /// <summary>
481        /// <para>
482        /// The last free link.
483        /// </para>
484        /// </summary>
485        public TLinkAddress LastFreeLink;
486
487        /// <summary>
488        /// <para>
489        /// The first allocated link.
490        /// </para>
491        /// </summary>
492        public TLinkAddress FirstAllocatedLink;
493
494        /// <summary>
495        /// <para>
496        /// The last allocated link.
497        /// </para>
498        /// </summary>
499        public TLinkAddress LastAllocatedLink;
500
501        /// <summary>
502        /// <para>
503        /// The first reserved link.
504        /// </para>
505        /// </summary>
506        public TLinkAddress FirstReservedLink;
507
508        /// <summary>
509        /// <para>
510        /// The last reserved link.
511        /// </para>
512        /// </summary>
513        public TLinkAddress LastReservedLink;
514
515        /// <summary>
516        /// <para>
517        /// The first free link.
518        /// </para>
519        /// </summary>
520        public TLinkAddress FirstFreeLink;
521
522        /// <summary>
523        /// <para>
524        /// The last free link.
525        /// </para>
526        /// </summary>
527        public TLinkAddress LastFreeLink;
528
529        /// <summary>
530        /// <para>
531        /// The first allocated link.
532        /// </para>
533        /// </summary>
534        public TLinkAddress FirstAllocatedLink;
535
536        /// <summary>
537        /// <para>
538        /// The last allocated link.
539        /// </para>
540        /// </summary>
541        public TLinkAddress LastAllocatedLink;
542
543        /// <summary>
544        /// <para>
545        /// The first reserved link.
546        /// </para>
547        /// </summary>
548        public TLinkAddress FirstReservedLink;
549
550        /// <summary>
551        /// <para>
552        /// The last reserved link.
553        /// </para>
554        /// </summary>
555        public TLinkAddress LastReservedLink;
556
557        /// <summary>
558        /// <para>
559        /// The first free link.
560        /// </para>
561        /// </summary>
562        public TLinkAddress FirstFreeLink;
563
564        /// <summary>
565        /// <para>
566        /// The last free link.
567        /// </para>
568        /// </summary>
569        public TLinkAddress LastFreeLink;
570
571        /// <summary>
572        /// <para>
573        /// The first allocated link.
574        /// </para>
575        /// </summary>
576        public TLinkAddress FirstAllocatedLink;
577
578        /// <summary>
579        /// <para>
580        /// The last allocated link.
581        /// </para>
582        /// </summary>
583        public TLinkAddress LastAllocatedLink;
584
585        /// <summary>
586        /// <para>
587        /// The first reserved link.
588        /// </para>
589        /// </summary>
590        public TLinkAddress FirstReservedLink;
591
592        /// <summary>
593        /// <para>
594        /// The last reserved link.
595        /// </para>
596        /// </summary>
597        public TLinkAddress LastReservedLink;
598
599        /// <summary>
600        /// <para>
601        /// The first free link.
602        /// </para>
603        /// </summary>
604        public TLinkAddress FirstFreeLink;
605
606        /// <summary>
607        /// <para>
608        /// The last free link.
609        /// </para>
610        /// </summary>
611        public TLinkAddress LastFreeLink;
612
613        /// <summary>
614        /// <para>
615        /// The first allocated link.
616        /// </para>
617        /// </summary>
618        public TLinkAddress FirstAllocatedLink;
619
620        /// <summary>
621        /// <para>
622        /// The last allocated link.
623        /// </para>
624        /// </summary>
625        public TLinkAddress LastAllocatedLink;
626
627        /// <summary>
628        /// <para>
629        /// The first reserved link.
630        /// </para>
631        /// </summary>
632        public TLinkAddress FirstReservedLink;
633
634        /// <summary>
635        /// <para>
636        /// The last reserved link.
637        /// </para>
638        /// </summary>
639        public TLinkAddress LastReservedLink;
640
641        /// <summary>
642        /// <para>
643        /// The first free link.
644        /// </para>
645        /// </summary>
646        public TLinkAddress FirstFreeLink;
647
648        /// <summary>
649        /// <para>
650        /// The last free link.
651        /// </para>
652        /// </summary>
653        public TLinkAddress LastFreeLink;
654
655        /// <summary>
656        /// <para>
657        /// The first allocated link.
658        /// </para>
659        /// </summary>
660        public TLinkAddress FirstAllocatedLink;
661
662        /// <summary>
663        /// <para>
664        /// The last allocated link.
665        /// </para>
666        /// </summary>
667        public TLinkAddress LastAllocatedLink;
668
669        /// <summary>
670        /// <para>
671        /// The first reserved link.
672        /// </para>
673        /// </summary>
674        public TLinkAddress FirstReservedLink;
675
676        /// <summary>
677        /// <para>
678        /// The last reserved link.
679        /// </para>
680        /// </summary>
681        public TLinkAddress LastReservedLink;
682
683        /// <summary>
684        /// <para>
685        /// The first free link.
686        /// </para>
687        /// </summary>
688        public TLinkAddress FirstFreeLink;
689
690        /// <summary>
691        /// <para>
692        /// The last free link.
693        /// </para>
694        /// </summary>
695        public TLinkAddress LastFreeLink;
696
697        /// <summary>
698        /// <para>
699        /// The first allocated link.
700        /// </para>
701        /// </summary>
702        public TLinkAddress FirstAllocatedLink;
703
704        /// <summary>
705        /// <para>
706        /// The last allocated link.
707        /// </para>
708        /// </summary>
709        public TLinkAddress LastAllocatedLink;
710
711        /// <summary>
712        /// <para>
713        /// The first reserved link.
714        /// </para>
715        /// </summary>
716        public TLinkAddress FirstReservedLink;
717
718        /// <summary>
719        /// <para>
720        /// The last reserved link.
721        /// </para>
722        /// </summary>
723        public TLinkAddress LastReservedLink;
724
725        /// <summary>
726        /// <para>
727        /// The first free link.
728        /// </para>
729        /// </summary>
730        public TLinkAddress FirstFreeLink;
731
732        /// <summary>
733        /// <para>
734        /// The last free link.
735        /// </para>
736        /// </summary>
737        public TLinkAddress LastFreeLink;
738
739        /// <summary>
740        /// <para>
741        /// The first allocated link.
742        /// </para>
743        /// </summary>
744        public TLinkAddress FirstAllocatedLink;
745
746        /// <summary>
747        /// <para>
748        /// The last allocated link.
749        /// </para>
750        /// </summary>
751        public TLinkAddress LastAllocatedLink;
752
753        /// <summary>
754        /// <para>
755        /// The first reserved link.
756        /// </para>
757        /// </summary>
758        public TLinkAddress FirstReservedLink;
759
760        /// <summary>
761        /// <para>
762        /// The last reserved link.
763        /// </para>
764        /// </summary>
765        public TLinkAddress LastReservedLink;
766
767        /// <summary>
768        /// <para>
769        /// The first free link.
770        /// </para>
771        /// </summary>
772        public TLinkAddress FirstFreeLink;
773
774        /// <summary>
775        /// <para>
776        /// The last free link.
777        /// </para>
778        /// </summary>
779        public TLinkAddress LastFreeLink;
780
781        /// <summary>
782        /// <para>
783        /// The first allocated link.
784        /// </para>
785        /// </summary>
786        public TLinkAddress FirstAllocatedLink;
787
788        /// <summary>
789        /// <para>
790        /// The last allocated link.
791        /// </para>
792        /// </summary>
793        public TLinkAddress LastAllocatedLink;
794
795        /// <summary>
796        /// <para>
797        /// The first reserved link.
798        /// </para>
799        /// </summary>
800        public TLinkAddress FirstReservedLink;
801
802        /// <summary>
803        /// <para>
804        /// The last reserved link.
805        /// </para>
806        /// </summary>
807        public TLinkAddress LastReservedLink;
808
809        /// <summary>
810        /// <para>
811        /// The first free link.
812        /// </para>
813        /// </summary>
814        public TLinkAddress FirstFreeLink;
815
816        /// <summary>
817        /// <para>
818        /// The last free link.
819        /// </para>
820        /// </summary>
821        public TLinkAddress LastFreeLink;
822
823        /// <summary>
824        /// <para>
825        /// The first allocated link.
826        /// </para>
827        /// </summary>
828        public TLinkAddress FirstAllocatedLink;
829
830        /// <summary>
831        /// <para>
832        /// The last allocated link.
833        /// </para>
834        /// </summary>
835        public TLinkAddress LastAllocatedLink;
836
837        /// <summary>
838        /// <para>
839        /// The first reserved link.
840        /// </para>
841        /// </summary>
842        public TLinkAddress FirstReservedLink;
843
844        /// <summary>
845        /// <para>
846        /// The last reserved link.
847        /// </para>
848        /// </summary>
849        public TLinkAddress LastReservedLink;
850
851        /// <summary>
852        /// <para>
853        /// The first free link.
854        /// </para>
855        /// </summary>
856        public TLinkAddress FirstFreeLink;
857
858        /// <summary>
859        /// <para>
860        /// The last free link.
861        /// </para>
862        /// </summary>
863        public TLinkAddress LastFreeLink;
864
865        /// <summary>
866        /// <para>
867        /// The first allocated link.
868        /// </para>
869        /// </summary>
870        public TLinkAddress FirstAllocatedLink;
871
872        /// <summary>
873        /// <para>
874        /// The last allocated link.
875        /// </para>
876        /// </summary>
877        public TLinkAddress LastAllocatedLink;
878
879        /// <summary>
880        /// <para>
881        /// The first reserved link.
882        /// </para>
883        /// </summary>
884        public TLinkAddress FirstReservedLink;
885
886        /// <summary>
887        /// <para>
888        /// The last reserved link.
889        /// </para>
890        /// </summary>
891        public TLinkAddress LastReservedLink;
892
893        /// <summary>
894        /// <para>
895        /// The first free link.
896        /// </para>
897        /// </summary>
898        public TLinkAddress FirstFreeLink;
899
900        /// <summary>
901        /// <para>
902        /// The last free link.
903        /// </para>
904        /// </summary>
905        public TLinkAddress LastFreeLink;
906
907        /// <summary>
908        /// <para>
909        /// The first allocated link.
910        /// </para>
911        /// </summary>
912        public TLinkAddress FirstAllocatedLink;
913
914        /// <summary>
915        /// <para>
916        /// The last allocated link.
917        /// </para>
918        /// </summary>
919        public TLinkAddress LastAllocatedLink;
920
921        /// <summary>
922        /// <para>
923        /// The first reserved link.
924        /// </para>
925        /// </summary>
926        public TLinkAddress FirstReservedLink;
927
928        /// <summary>
929        /// <para>
930        /// The last reserved link.
931        /// </para>
932        /// </summary>
933        public TLinkAddress LastReservedLink;
934
935        /// <summary>
936        /// <para>
937        /// The first free link.
938        /// </para>
939        /// </summary>
940        public TLinkAddress FirstFreeLink;
941
942        /// <summary>
943        /// <para>
944        /// The last free link.
945        /// </para>
946        /// </summary>
947        public TLinkAddress LastFreeLink;
948
949        /// <summary>
950        /// <para>
951        /// The first allocated link.
952        /// </para>
953        /// </summary>
954        public TLinkAddress FirstAllocatedLink;
955
956        /// <summary>
957        /// <para>
958        /// The last allocated link.
959        /// </para>
960        /// </summary>
961        public TLinkAddress LastAllocatedLink;
962
963        /// <summary>
964        /// <para>
965        /// The first reserved link.
966        /// </para>
967        /// </summary>
968        public TLinkAddress FirstReservedLink;
969
970        /// <summary>
971        /// <para>
972        /// The last reserved link.
973        /// </para>
974        /// </summary>
975        public TLinkAddress LastReservedLink;
976
977        /// <summary>
978        /// <para>
979        /// The first free link.
980        /// </para>
981        /// </summary>
982        public TLinkAddress FirstFreeLink;
983
984        /// <summary>
985        /// <para>
986        /// The last free link.
987        /// </para>
988        /// </summary>
989        public TLinkAddress LastFreeLink;
990
991        /// <summary>
992        /// <para>
993        /// The first allocated link.
994        /// </para>
995        /// </summary>
996        public TLinkAddress FirstAllocatedLink;
997
998        /// <summary>
999        /// <para>
1000        /// The last allocated link.
1001        /// </para>
1002        /// </summary>
1003        public TLinkAddress LastAllocatedLink;
1004
1005        /// <summary>
1006        /// <para>
1007        /// The first reserved link.
1008        /// </para>
1009        /// </summary>
1010        public TLinkAddress FirstReservedLink;
1011
1012        /// <summary>
1013        /// <para>
1014        /// The last reserved link.
1015        /// </para>
1016        /// </summary>
1017        public TLinkAddress LastReservedLink;
1018
1019        /// <summary>
1020        /// <para>
1021        /// The first free link.
1022        /// </para>
1023        /// </summary>
1024        public TLinkAddress FirstFreeLink;
1025
1026        /// <summary>
1027        /// <para>
1028        /// The last free link.
1029        /// </para>
1030        /// </summary>
1031        public TLinkAddress LastFreeLink;
1032
1033        /// <summary>
1034        /// <para>
1035        /// The first allocated link.
1036        /// </para>
1037        /// </summary>
1038        public TLinkAddress FirstAllocatedLink;
1039
1040        /// <summary>
1041        /// <para>
1042        /// The last allocated link.
1043        /// </para>
1044        /// </summary>
1045        public TLinkAddress LastAllocatedLink;
1046
1047        /// <summary>
1048        /// <para>
1049        /// The first reserved link.
1050        /// </para>
1051        /// </summary>
1052        public TLinkAddress FirstReservedLink;
1053
1054        /// <summary>
1055        /// <para>
1056        /// The last reserved link.
1057        /// </para>
1058        /// </summary>
1059        public TLinkAddress LastReservedLink;
1060
1061        /// <summary>
1062        /// <para>
1063        /// The first free link.
1064        /// </para>
1065        /// </summary>
1066        public TLinkAddress FirstFreeLink;
1067
1068        /// <summary>
1069        /// <para>
1070        /// The last free link.
1071        /// </para>
1072        /// </summary>
1073        public TLinkAddress LastFreeLink;
1074
1075        /// <summary>
1076        /// <para>
1077        /// The first allocated link.
1078        /// </para>
1079        /// </summary>
1080        public TLinkAddress FirstAllocatedLink;
1081
1082        /// <summary>
1083        /// <para>
1084        /// The last allocated link.
1085        /// </para>
1086        /// </summary>
1087        public TLinkAddress LastAllocatedLink;
1088
1089        /// <summary>
1090        /// <para>
1091        /// The first reserved link.
1092        /// </para>
1093        /// </summary>
1094        public TLink
```



```

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>
108    /// </summary>
109    /// <param name="other">
110    /// <para>The other.</para>
111    /// <para></para>
112    /// </param>
113    /// <returns>
114    /// <para>The bool</para>
115    /// <para></para>
116    /// </returns>
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    public bool Equals(LinksHeader<TLinkAddress> other)
119        => _equalityComparer.Equals(AllocatedLinks, other.AllocatedLinks)
120        && _equalityComparer.Equals(ReservedLinks, other.ReservedLinks)
121        && _equalityComparer.Equals(FreeLinks, other.FreeLinks)
122        && _equalityComparer.Equals(FirstFreeLink, other.FirstFreeLink)
123        && _equalityComparer.Equals(RootAsSource, other.RootAsSource)
124        && _equalityComparer.Equals(RootAsTarget, other.RootAsTarget)
125        && _equalityComparer.Equals(LastFreeLink, other.LastFreeLink)
126        && _equalityComparer.Equals(Reserved8, other.Reserved8);
127
128    /// <summary>
129    /// <para>
130    /// Gets the hash code.

```

```

/// </para>
/// <para></para>
/// </summary>
/// <returns>
/// <para>The int</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public override int GetHashCode() => (AllocatedLinks, ReservedLinks, FreeLinks,
↪ FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();

[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool operator ==(LinksHeader<TLinkAddress> left, LinksHeader<TLinkAddress>
↪ right) => left.Equals(right);

[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static bool operator !=(LinksHeader<TLinkAddress> left, LinksHeader<TLinkAddress>
↪ right) => !(left == right);

```

1.33 `./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethod`

```

using System;
using System.Text;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Methods.Trees;
using Platform.Converters;
using Platform.Delegates;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the external links recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLinkAddress}"/>
    /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
    public unsafe abstract class
    ↪ ExternallinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress> :
    ↪ RecursionlessSizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
    {
        private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
        ↪ = UncheckedConverter<TLinkAddress, long>.Default;

        /// <summary>
        /// <para>
        /// The break.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLinkAddress Break;

        /// <summary>
        /// <para>
        /// The continue.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLinkAddress Continue;

        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksDataParts;

        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksIndexParts;

        /// <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>
111    /// <para>
112    /// Gets the base part value using the specified link.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="link">
117    /// <para>The link.</para>
118    /// <para></para>
119    /// </param>
120    /// <returns>
121    /// <para>The link</para>
122    /// <para></para>
123    /// </returns>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected abstract TLinkAddress GetBasePartValue(TLinkAddress link);
126
127    /// <summary>
128    /// <para>
129    /// Determines whether this instance first is to the right of second.
130    /// </para>
131    /// <para></para>
132    /// </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
195    /// <summary>
196    /// <para>
197    /// Gets the link data part reference using the specified link.
198    /// </para>
199    /// <para></para>
200    /// </summary>
201    /// <param name="link">
202    /// <para>The link.</para>
203    /// <para></para>
204    /// </param>

```

```

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

```

```

274 }
275
276 /// <summary>
277 /// <para>
278 /// Determines whether this instance first is to the right of second.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 /// <param name="first">
283 /// <para>The first.</para>
284 /// <para></para>
285 /// </param>
286 /// <param name="second">
287 /// <para>The second.</para>
288 /// <para></para>
289 /// </param>
290 /// <returns>
291 /// <para>The bool</para>
292 /// <para></para>
293 /// </returns>
294 [MethodImpl(MethodImplOptions.AggressiveInlining)]
295 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
    ↪ second)
296 {
297     ref var firstLink = ref GetLinkDataPartReference(first);
298     ref var secondLink = ref GetLinkDataPartReference(second);
299     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
300 }
301
302 /// <summary>
303 /// <para>
304 /// The zero.
305 /// </para>
306 /// <para></para>
307 /// </summary>
308 public TLinkAddress this[TLinkAddress index]
309 {
310     [MethodImpl(MethodImplOptions.AggressiveInlining)]
311     get
312     {
313         var root = GetTreeRoot();
314         if (GreaterOrEqualThan(index, GetSize(root)))
315         {
316             return Zero;
317         }
318         while (!EqualToZero(root))
319         {
320             var left = GetLeftOrDefault(root);
321             var leftSize = GetSizeOrZero(left);
322             if (LessThan(index, leftSize))
323             {
324                 root = left;
325                 continue;
326             }
327             if (AreEqual(index, leftSize))
328             {
329                 return root;
330             }
331             root = GetRightOrDefault(root);
332             index = Subtract(index, Increment(leftSize));
333         }
334         return Zero; // TODO: Impossible situation exception (only if tree structure
    ↪ broken)
335     }
336 }
337
338 /// <summary>
339 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
340 /// </summary>
341 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
342 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
343 /// <returns>Индекс искомой связи.</returns>
344 [MethodImpl(MethodImplOptions.AggressiveInlining)]
345 public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
346 {
347     var root = GetTreeRoot();

```

```

348 while (!EqualToZero(root))
349 {
350     ref var rootLink = ref GetLinkDataPartReference(root);
351     var rootSource = rootLink.Source;
352     var rootTarget = rootLink.Target;
353     if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
354         ↳ node.Key < root.Key
355     {
356         root = GetLeftOrDefault(root);
357     }
358     else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
359         ↳ node.Key > root.Key
360     {
361         root = GetRightOrDefault(root);
362     }
363     else // node.Key == root.Key
364     {
365         return root;
366     }
367 }
368 return Zero;
369 }
370
371 // TODO: Return indices range instead of references count
372 /// <summary>
373 /// <para>
374 /// Counts the usages using the specified link.
375 /// </para>
376 /// <para></para>
377 /// </summary>
378 /// <param name="link">
379 /// <para>The link.</para>
380 /// <para></para>
381 /// </param>
382 /// <returns>
383 /// <para>The link</para>
384 /// <para></para>
385 /// </returns>
386 [MethodImpl(MethodImplOptions.AggressiveInlining)]
387 public TLinkAddress CountUsages(TLinkAddress link)
388 {
389     var root = GetTreeRoot();
390     var total = GetSize(root);
391     var totalRightIgnore = Zero;
392     while (!EqualToZero(root))
393     {
394         var @base = GetBasePartValue(root);
395         if (LessOrEqualThan(@base, link))
396         {
397             root = GetRightOrDefault(root);
398         }
399         else
400         {
401             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
402             root = GetLeftOrDefault(root);
403         }
404     }
405     root = GetTreeRoot();
406     var totalLeftIgnore = Zero;
407     while (!EqualToZero(root))
408     {
409         var @base = GetBasePartValue(root);
410         if (GreaterOrEqualThan(@base, link))
411         {
412             root = GetLeftOrDefault(root);
413         }
414         else
415         {
416             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
417             root = GetRightOrDefault(root);
418         }
419     }
420     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
421 }
422
423 /// <summary>
424 /// <para>
425 /// 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) =>
    ↳ EachUsageCore(@base, GetTreeRoot(), handler);
441
442 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
    ↳ low-level MSIL stack.
443 [MethodImpl(MethodImplOptions.AggressiveInlining)]
444 private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
    ↳ ReadHandler<TLinkAddress>? handler)
445 {
446     var @continue = Continue;
447     if (EqualToZero(link))
448     {
449         return @continue;
450     }
451     var linkBasePart = GetBasePartValue(link);
452     var @break = Break;
453     if (GreaterThan(linkBasePart, @base))
454     {
455         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
456         {
457             return @break;
458         }
459     }
460     else if (LessThan(linkBasePart, @base))
461     {
462         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
463         {
464             return @break;
465         }
466     }
467     else //if (linkBasePart == @base)
468     {
469         if (AreEqual(handler(GetLinkValues(link)), @break))
470         {
471             return @break;
472         }
473         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
474         {
475             return @break;
476         }
477         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
478         {
479             return @break;
480         }
481     }
482     return @continue;
483 }
484
485 /// <summary>
486 /// <para>
487 /// Prints the node value using the specified node.
488 /// </para>
489 /// <para></para>
490 /// </summary>
491 /// <param name="node">
492 /// <para>The node.</para>
493 /// <para></para>
494 /// </param>
495 /// <param name="sb">
496 /// <para>The sb.</para>
497 /// <para></para>
498 /// </param>

```



```

499     [MethodImpl(MethodImplOptions.AggressiveInlining)]
500     protected override void PrintNodeValue(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/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;
43
44         /// <summary>
45         /// <para>
46         /// The links data parts.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly byte* LinksDataParts;
51
52         /// <summary>
53         /// <para>
54         /// The links index parts.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         protected readonly byte* LinksIndexParts;
59
60         /// <summary>
61         /// <para>
62         /// The header.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         protected readonly byte* Header;
67
68         /// <summary>
69         /// <para>
70

```

```

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

```

```

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

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

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

```

```

212     /// <summary>
213     /// <para>
214     /// Gets the link index part reference using the specified link.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="link">
219     /// <para>The link.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>A ref raw link index part of t link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected virtual ref RawLinkIndexPart<TLinkAddress>
        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
        ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
        ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
        ↪ _addressToInt64Converter.Convert(link)));

228
229     /// <summary>
230     /// <para>
231     /// Gets the link values using the specified link index.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="linkIndex">
236     /// <para>The link index.</para>
237     /// <para></para>
238     /// </param>
239     /// <returns>
240     /// <para>A list of t link</para>
241     /// <para></para>
242     /// </returns>
243     [MethodImpl(MethodImplOptions.AggressiveInlining)]
244     protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
245     {
246         ref var link = ref GetLinkDataPartReference(linkIndex);
247         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
248     }
249
250     /// <summary>
251     /// <para>
252     /// Determines whether this instance first is to the left of second.
253     /// </para>
254     /// <para></para>
255     /// </summary>
256     /// <param name="first">
257     /// <para>The first.</para>
258     /// <para></para>
259     /// </param>
260     /// <param name="second">
261     /// <para>The second.</para>
262     /// <para></para>
263     /// </param>
264     /// <returns>
265     /// <para>The bool</para>
266     /// <para></para>
267     /// </returns>
268     [MethodImpl(MethodImplOptions.AggressiveInlining)]
269     protected override bool 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,
        ↪ secondLink.Source, secondLink.Target);
274     }
275
276     /// <summary>
277     /// <para>
278     /// Determines whether this instance first is to the right of second.
279     /// </para>
280     /// <para></para>
281     /// </summary>
282     /// <param name="first">
283     /// <para>The first.</para>
284     /// <para></para>

```

```

285     /// </param>
286     /// <param name="second">
287     /// <para>The second.</para>
288     /// <para></para>
289     /// </param>
290     /// <returns>
291     /// <para>The bool</para>
292     /// <para></para>
293     /// </returns>
294     [MethodImpl(MethodImplOptions.AggressiveInlining)]
295     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
    ↪ second)
296     {
297         ref var firstLink = ref GetLinkDataPartReference(first);
298         ref var secondLink = ref GetLinkDataPartReference(second);
299         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
300     }
301
302     /// <summary>
303     /// <para>
304     /// The zero.
305     /// </para>
306     /// <para></para>
307     /// </summary>
308     public TLinkAddress this[TLinkAddress index]
309     {
310         [MethodImpl(MethodImplOptions.AggressiveInlining)]
311         get
312         {
313             var root = GetTreeRoot();
314             if (GreaterOrEqualThan(index, GetSize(root)))
315             {
316                 return Zero;
317             }
318             while (!EqualToZero(root))
319             {
320                 var left = GetLeftOrDefault(root);
321                 var leftSize = GetSizeOrZero(left);
322                 if (LessThan(index, leftSize))
323                 {
324                     root = left;
325                     continue;
326                 }
327                 if (AreEqual(index, leftSize))
328                 {
329                     return root;
330                 }
331                 root = GetRightOrDefault(root);
332                 index = Subtract(index, Increment(leftSize));
333             }
334             return Zero; // TODO: Impossible situation exception (only if tree structure
    ↪ broken)
335         }
336     }
337
338     /// <summary>
339     /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
340     /// </summary>
341     /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
342     /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
343     /// <returns>Индекс искомой связи.</returns>
344     [MethodImpl(MethodImplOptions.AggressiveInlining)]
345     public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
346     {
347         var root = GetTreeRoot();
348         while (!EqualToZero(root))
349         {
350             ref var rootLink = ref GetLinkDataPartReference(root);
351             var rootSource = rootLink.Source;
352             var rootTarget = rootLink.Target;
353             if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
    ↪ node.Key < root.Key
354             {
355                 root = GetLeftOrDefault(root);
356             }

```

```

357         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
358             ↪ node.Key > root.Key
359         {
360             root = GetRightOrDefault(root);
361         }
362         else // node.Key == root.Key
363         {
364             return root;
365         }
366     }
367     return Zero;
368 }
369
370 // TODO: Return indices range instead of references count
371 /// <summary>
372 /// <para>
373 /// Counts the usages using the specified link.
374 /// </para>
375 /// <para></para>
376 /// </summary>
377 /// <param name="link">
378 /// <para>The link.</para>
379 /// <para></para>
380 /// </param>
381 /// <returns>
382 /// <para>The link</para>
383 /// <para></para>
384 /// </returns>
385 [MethodImpl(MethodImplOptions.AggressiveInlining)]
386 public TLinkAddress CountUsages(TLinkAddress link)
387 {
388     var root = GetTreeRoot();
389     var total = GetSize(root);
390     var totalRightIgnore = Zero;
391     while (!EqualToZero(root))
392     {
393         var @base = GetBasePartValue(root);
394         if (LessOrEqualThan(@base, link))
395         {
396             root = GetRightOrDefault(root);
397         }
398         else
399         {
400             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
401             root = GetLeftOrDefault(root);
402         }
403     }
404     root = GetTreeRoot();
405     var totalLeftIgnore = Zero;
406     while (!EqualToZero(root))
407     {
408         var @base = GetBasePartValue(root);
409         if (GreaterOrEqualThan(@base, link))
410         {
411             root = GetLeftOrDefault(root);
412         }
413         else
414         {
415             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
416             root = GetRightOrDefault(root);
417         }
418     }
419     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
420 }
421
422 /// <summary>
423 /// <para>
424 /// Eaches the usage using the specified base.
425 /// </para>
426 /// <para></para>
427 /// </summary>
428 /// <param name="@base">
429 /// <para>The base.</para>
430 /// <para></para>
431 /// </param>
432 /// <param name="handler">
433 /// <para>The handler.</para>
434 /// <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.
491 /// </para>
492 /// <para></para>
493 /// </summary>
494 /// <param name="node">
495 /// <para>The node.</para>
496 /// <para></para>
497 /// </param>
498 /// <param name="sb">
499 /// <para>The sb.</para>
500 /// <para></para>
501 /// </param>
502 [MethodImpl(MethodImplOptions.AggressiveInlining)]
503 protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
504 {
505     ref var link = ref GetLinkDataPartReference(node);
506     sb.Append(' ');
507     sb.Append(link.Source);
508     sb.Append('-');
509     sb.Append('>');
510     sb.Append(link.Target);
511 }

```

```
509     }
510 }
```

1.35 ./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>
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).LeftAsSource;
62
63         /// <summary>
64         /// <para>
65         /// Gets the right reference using the specified node.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         /// <param name="node">
70         /// <para>The node.</para>
71         /// <para></para>
72         /// </param>
73         /// <returns>
74         /// <para>The ref link</para>
75         /// <para></para>
76         /// </returns>
77     }
```



```

70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
74         ↪ GetLinkIndexPartReference(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     /// </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    /// </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).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    /// </param>
119    /// <param name="left">
120    /// <para>The left.</para>
121    /// </param>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
124        ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
125
126    /// <summary>
127    /// <para>
128    /// Sets the right using the specified node.
129    /// </para>
130    /// <para></para>
131    /// </summary>
132    /// <param name="node">
133    /// <para>The node.</para>
134    /// </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;

```

```

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

```

```

217     /// <param name="firstTarget">
218     /// <para>The first target.</para>
219     /// <para></para>
220     /// </param>
221     /// <param name="secondSource">
222     /// <para>The second source.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="secondTarget">
226     /// <para>The second target.</para>
227     /// <para></para>
228     /// </param>
229     /// <returns>
230     /// <para>The bool</para>
231     /// <para></para>
232     /// </returns>
233     [MethodImpl(MethodImplOptions.AggressiveInlining)]
234     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
        ↪ LessThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
        ↪ LessThan(firstTarget, secondTarget));

235     /// <summary>
236     /// <para>
237     /// <para>Determines whether this instance first is to the right of second.
238     /// </para>
239     /// <para></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 FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
        ↪ GreaterThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
        ↪ GreaterThan(firstTarget, secondTarget));

264     /// <summary>
265     /// <para>
266     /// <para>Clears the node using the specified node.
267     /// </para>
268     /// <para></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(TLinkAddress node)
276     {
277         ref var link = ref GetLinkIndexPartReference(node);
278         link.LeftAsSource = Zero;
279         link.RightAsSource = Zero;
280         link.SizeAsSource = Zero;
281     }
282 }
283 }
284 }

```

```

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)]
59        protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
60            ↳ GetLinkIndexPartReference(node).LeftAsSource;
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).RightAsSource;
79
80        /// <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">

```

```

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).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    /// <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).SizeAsSource = 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().RootAsSource;
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).Source;
206
207    /// <summary>
208    /// <para>
209    /// Determines whether this instance first is to the left of second.
210    /// </para>
211    /// </summary>
212    /// <param name="firstSource">
213    /// <para>The first source.</para>
214    /// <para></para>
215    /// </param>
216    /// <param name="firstTarget">
217    /// <para>The first target.</para>
218    /// <para></para>
219    /// </param>
220    /// <param name="secondSource">
221    /// <para>The second source.</para>
222    /// <para></para>
223    /// </param>
224    /// </summary>

```

```

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

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>
243     /// </param>
244     /// <param name="firstTarget">
245     /// <para>The first target.</para>
246     /// </para>
247     /// <param name="secondSource">
248     /// <para>The second source.</para>
249     /// </para>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// </para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// </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));

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

```

1.37 ./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> :
    ↳ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
15  {
16      /// <summary>
17      /// <para>
18      /// Initializes a new <see
    ↳ cref="ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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).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) =>
91        ↪ GetLinkIndexPartReference(node).LeftAsTarget;
92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// <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).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        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(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>

```

```

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     /// <para></para>
166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// </param>
170     /// <param name="size">
171     /// <para>The size.</para>
172     /// </param>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
175         ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
176
177     /// <summary>
178     /// <para>
179     /// Gets the tree root.
180     /// </para>
181     /// <para></para>
182     /// </summary>
183     /// <returns>
184     /// <para>The link</para>
185     /// <para></para>
186     /// </returns>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsTarget;
189
190     /// <summary>
191     /// <para>
192     /// Gets the base part value using the specified link.
193     /// </para>
194     /// <para></para>
195     /// </summary>
196     /// <param name="link">
197     /// <para>The link.</para>
198     /// </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     /// </param>
216     /// <param name="firstTarget">
217     /// <para>The first target.</para>
218     /// </param>
219     /// <param name="secondSource">
220     /// <para>The second source.</para>
221     /// </param>
222     /// <param name="secondTarget">
223     /// <para>The second target.</para>
224     /// </param>
225     /// <returns>
226     /// <para>The bool</para>
227     /// </returns>
228     /// <para></para>
229     /// </summary>
230     /// <para></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/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> :
        ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>
15    {

```

```

16    /// <summary>
17    /// <para>
18    /// Initializes a new <see cref="ExternalLinksTargetsSizeBalancedTreeMethods"/> instance.
19    /// </para>
20    /// <para></para>
21    /// </summary>
22    /// <param name="constants">
23    /// <para>A constants.</para>
24    /// <para></para>
25    /// </param>
26    /// <param name="linksDataParts">
27    /// <para>A links data parts.</para>
28    /// <para></para>
29    /// </param>
30    /// <param name="linksIndexParts">
31    /// <para>A links index parts.</para>
32    /// <para></para>
33    /// </param>
34    /// <param name="header">
35    /// <para>A header.</para>
36    /// <para></para>
37    /// </param>
38    [MethodImpl(MethodImplOptions.AggressiveInlining)]
39    public ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<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).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) =>
    ↳ GetLinkIndexPartReference(node).RightAsTarget;
108
109 /// <summary>
110 /// <para>
111 /// Sets the left using the specified node.
112 /// </para>
113 /// <para></para>
114 /// </summary>
115 /// <param name="node">
116 /// <para>The node.</para>
117 /// <para></para>
118 /// </param>
119 /// <param name="left">
120 /// <para>The left.</para>
121 /// <para></para>
122 /// </param>
123 [MethodImpl(MethodImplOptions.AggressiveInlining)]
124 protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
    ↳ GetLinkIndexPartReference(node).LeftAsTarget = left;
125
126 /// <summary>
127 /// <para>
128 /// Sets the right using the specified node.
129 /// </para>
130 /// <para></para>
131 /// </summary>
132 /// <param name="node">
133 /// <para>The node.</para>
134 /// <para></para>
135 /// </param>
136 /// <param name="right">
137 /// <para>The right.</para>
138 /// <para></para>
139 /// </param>
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
    ↳ GetLinkIndexPartReference(node).RightAsTarget = right;
142
143 /// <summary>
144 /// <para>
145 /// Gets the size using the specified node.
146 /// </para>
147 /// <para></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) =>
    ↳ GetLinkIndexPartReference(node).SizeAsTarget;
159
160 /// <summary>

```

```

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

```

```

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 }

```

1.39 ./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="ILinksTreeMethods{TLinkAddress}"/>
22  public unsafe abstract class
    ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress> :
    ↳ RecursionlessSizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
23  {
24      private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
    ↳ = UncheckedConverter<TLinkAddress, long>.Default;
25
26      /// <summary>
27      /// <para>
28      /// 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="InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
65      /// </para>
66      /// <para></para>
67      /// </summary>
68      /// <param name="constants">
69      /// <para>A constants.</para>
70      /// <para></para>
71      /// </param>
72      /// <param name="linksDataParts">
73      /// <para>A links data parts.</para>
74      /// <para></para>
75      /// </param>
76      /// <param name="linksIndexParts">
77      /// <para>A links index parts.</para>
78      /// <para></para>
79      /// </param>
80      /// <param name="header">
81      /// <para>A header.</para>
82      /// <para></para>
83      /// </param>
84      [MethodImpl(MethodImplOptions.AggressiveInlining)]
85      protected
    ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
    ↳ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
86      {
87          LinksDataParts = linksDataParts;
88          LinksIndexParts = linksIndexParts;
89          Header = header;
90          Break = constants.Break;
91          Continue = constants.Continue;
92      }
93

```



```

94     /// <summary>
95     /// <para>
96     /// Gets the tree root using the specified link.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="link">
101    /// <para>The link.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected abstract 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>

```

```

168    /// <param name="link">
169    /// <para>The link.</para>
170    /// <para></para>
171    /// </param>
172    /// <returns>
173    /// <para>A ref raw link index part of t link</para>
174    /// <para></para>
175    /// </returns>
176    [MethodImpl(MethodImplOptions.AggressiveInlining)]
177    protected virtual ref RawLinkIndexPart<TLinkAddress>
178    ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
179    ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
180    ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
181    ↪ _addressToInt64Converter.Convert(link)));
182
183    /// <summary>
184    /// <para>
185    /// Determines whether this instance first is to the left of second.
186    /// </para>
187    /// <para></para>
188    /// </summary>
189    /// <param name="first">
190    /// <para>The first.</para>
191    /// <para></para>
192    /// </param>
193    /// <param name="second">
194    /// <para>The second.</para>
195    /// <para></para>
196    /// </param>
197    /// <returns>
198    /// <para>The bool</para>
199    /// <para></para>
200    /// </returns>
201    [MethodImpl(MethodImplOptions.AggressiveInlining)]
202    protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress
203    ↪ second) => LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
204
205    /// <summary>
206    /// <para>
207    /// Determines whether this instance first is to the right of second.
208    /// </para>
209    /// <para></para>
210    /// </summary>
211    /// <param name="first">
212    /// <para>The first.</para>
213    /// <para></para>
214    /// </param>
215    /// <param name="second">
216    /// <para>The second.</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 FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
225    ↪ second) => GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
226
227    /// <summary>
228    /// <para>
229    /// Gets the link values using the specified link index.
230    /// </para>
231    /// <para></para>
232    /// </summary>
233    /// <param name="linkIndex">
234    /// <para>The link index.</para>
235    /// <para></para>
236    /// </param>
237    /// <returns>
238    /// <para>A list of t link</para>
239    /// <para></para>
240    /// </returns>
241    [MethodImpl(MethodImplOptions.AggressiveInlining)]
242    protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
243    {
244        ref var link = ref GetLinkDataPartReference(linkIndex);
245        return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
246    }

```

```

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         var rootKey = GetKeyPartValue(root);
313         if (LessThan(key, rootKey)) // node.Key < root.Key
314         {
315             root = GetLeftOrDefault(root);
316         }
317         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)
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 }

```

```

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 }

```

1.40 ./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.
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 data parts.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly byte* LinksDataParts;
51
52         /// <summary>
53         /// <para>
54         /// The links index parts.
55         /// </para>
56         /// </summary>
57         protected readonly byte* LinksIndexParts;
58     }
59 }

```

```

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
95     /// <summary>
96     /// <para>
97     /// Gets the tree root 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 GetTreeRoot(TLinkAddress link);
111
112    /// <summary>
113    /// <para>
114    /// Gets the base part value using the specified link.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="link">
119    /// <para>The link.</para>
120    /// <para></para>
121    /// </param>
122    /// <returns>
123    /// <para>The link</para>
124    /// <para></para>
125    /// </returns>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    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)]
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>

```

```

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
    ↪ 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)
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>
403     /// </param>
404     /// <param name="sb">
405     /// <para>The sb.</para>
406     /// <para></para>
407     /// </param>
408     [MethodImpl(MethodImplOptions.AggressiveInlining)]
409     protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
410     {
411         ref var link = ref GetLinkDataPartReference(node);
412         sb.Append(' ');
413         sb.Append(link.Source);
414         sb.Append('-');
415         sb.Append('>');
416         sb.Append(link.Target);
417     }
418 }

```

1.41 ./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         /// <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         /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         /// <param name="constants">
48         /// <para>A constants.</para>
49         /// <para></para>
50         /// </param>
51         /// <param name="linksDataParts">
52         /// <para>A links data parts.</para>
53         /// <para></para>
54         /// </param>
55         /// <param name="linksIndexParts">
56         /// <para>A links index parts.</para>
57         /// <para></para>
58         /// </param>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress> constants,
61             ↳ byte* linksDataParts, byte* linksIndexParts)
62         {
63             _linksDataParts = linksDataParts;
64             _linksIndexParts = linksIndexParts;
65             Break = constants.Break;
66             Continue = constants.Continue;
67         }
68         /// <summary>
69         /// <para>
70         /// Gets the link data part reference using the specified link.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <param name="link">
75         /// <para>The link.</para>
76         /// <para></para>
77         /// </param>
78         /// <returns>
79         /// <para>A ref raw link data part of t link</para>
80         /// <para></para>
81         /// </returns>
82         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

82     protected virtual ref RawLinkDataPart<TLinkAddress>
83     ↪ GetLinkDataPartReference(TLinkAddress link) => ref
84     ↪ AsRef<RawLinkDataPart<TLinkAddress>>(_linksDataParts +
85     ↪ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
86     ↪ _addressToInt64Converter.Convert(link)));
87
88     /// <summary>
89     /// <para>
90     /// Gets the link index part reference using the specified link.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="link">
95     /// <para>The link.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>A ref raw link index part of t link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected virtual ref RawLinkIndexPart<TLinkAddress>
104    ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
105    ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(_linksIndexParts +
106    ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
107    ↪ _addressToInt64Converter.Convert(link)));
108
109    /// <summary>
110    /// <para>
111    /// Gets the first using the specified head.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="head">
116    /// <para>The head.</para>
117    /// <para></para>
118    /// </param>
119    /// <returns>
120    /// <para>The link</para>
121    /// <para></para>
122    /// </returns>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override TLinkAddress GetFirst(TLinkAddress head) =>
125    ↪ GetLinkIndexPartReference(head).RootAsSource;
126
127    /// <summary>
128    /// <para>
129    /// Gets the last using the specified head.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="head">
134    /// <para>The head.</para>
135    /// <para></para>
136    /// </param>
137    /// <returns>
138    /// <para>The link</para>
139    /// <para></para>
140    /// </returns>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected override TLinkAddress GetLast(TLinkAddress head)
143    {
144        var first = GetLinkIndexPartReference(head).RootAsSource;
145        if (EqualToZero(first))
146        {
147            return first;
148        }
149        else
150        {
151            return GetPrevious(first);
152        }
153    }
154
155    /// <summary>
156    /// <para>
157    /// Gets the previous using the specified element.
158    /// </para>
159    /// <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>
205     /// </summary>
206     /// <param name="head">
207     /// <para>The head.</para>
208     /// <para></para>
209     /// </param>
210     /// <param name="element">
211     /// <para>The element.</para>
212     /// <para></para>
213     /// </param>
214     [MethodImpl(MethodImplOptions.AggressiveInlining)]
215     protected override void SetFirst(TLinkAddress head, TLinkAddress element) =>
216         ↪ GetLinkIndexPartReference(head).RootAsSource = element;
217
218     /// <summary>
219     /// <para>
220     /// Sets the last using the specified head.
221     /// </para>
222     /// <para></para>
223     /// </summary>
224     /// <param name="head">
225     /// <para>The head.</para>
226     /// <para></para>
227     /// </param>
228     /// <param name="element">

```

```

225 /// <para>The element.</para>
226 /// <para></para>
227 /// </param>
228 [MethodImpl(MethodImplOptions.AggressiveInlining)]
229 protected override void SetLast(TLinkAddress head, TLinkAddress element)
230 {
231     //var first = GetLinkIndexPartReference(head).RootAsSource;
232     //if (EqualToZero(first))
233     //{
234         //    SetFirst(head, element);
235     //}
236     //else
237     //{
238         //    SetPrevious(first, element);
239     //}
240 }
241
242 /// <summary>
243 /// <para>
244 /// Sets the previous using the specified element.
245 /// </para>
246 /// <para></para>
247 /// </summary>
248 /// <param name="element">
249 /// <para>The element.</para>
250 /// <para></para>
251 /// </param>
252 /// <param name="previous">
253 /// <para>The previous.</para>
254 /// <para></para>
255 /// </param>
256 [MethodImpl(MethodImplOptions.AggressiveInlining)]
257 protected override void SetPrevious(TLinkAddress element, TLinkAddress previous) =>
258     ↪ GetLinkIndexPartReference(element).LeftAsSource = previous;
259
260 /// <summary>
261 /// <para>
262 /// Sets the next using the specified element.
263 /// </para>
264 /// <para></para>
265 /// </summary>
266 /// <param name="element">
267 /// <para>The element.</para>
268 /// <para></para>
269 /// </param>
270 /// <param name="next">
271 /// <para>The next.</para>
272 /// <para></para>
273 /// </param>
274 [MethodImpl(MethodImplOptions.AggressiveInlining)]
275 protected override void SetNext(TLinkAddress element, TLinkAddress next) =>
276     ↪ GetLinkIndexPartReference(element).RightAsSource = next;
277
278 /// <summary>
279 /// <para>
280 /// Sets the size using the specified head.
281 /// </para>
282 /// <para></para>
283 /// </summary>
284 /// <param name="head">
285 /// <para>The head.</para>
286 /// <para></para>
287 /// </param>
288 /// <param name="size">
289 /// <para>The size.</para>
290 /// <para></para>
291 /// </param>
292 [MethodImpl(MethodImplOptions.AggressiveInlining)]
293 protected override void SetSize(TLinkAddress head, TLinkAddress size) =>
294     ↪ GetLinkIndexPartReference(head).SizeAsSource = size;
295
296 /// <summary>
297 /// <para>
298 /// Counts the usages using the specified head.
299 /// </para>
300 /// <para></para>
301 /// </summary>
302 /// <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 }

```

1.42 ./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<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).LeftAsSource;
62
63        /// <summary>
64        /// <para>
65        /// Gets the right reference using the specified node.
66        /// </para>
67        /// <para></para>
68        /// </summary>
69        /// <param name="node">
70        /// <para>The node.</para>
71        /// <para></para>
72        /// </param>
73        /// <returns>
74        /// <para>The ref link</para>
75        /// <para></para>
76        /// </returns>
77        [MethodImpl(MethodImplOptions.AggressiveInlining)]
78        protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
79            ↪ GetLinkIndexPartReference(node).RightAsSource;
80
81        /// <summary>
82        /// <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>

```

```

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).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     /// <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).SizeAsSource = size;
177
178     /// <summary>
179     /// <para>
180     /// Gets the tree root using the specified link.
181     /// </para>
182     /// <para></para>
183     /// </summary>
184     /// <param name="link">
185     /// <para>The link.</para>
186     /// <para></para>
187     /// </param>
188     /// <returns>
189     /// <para>The link</para>
190     /// <para></para>
191     /// </returns>
192     [MethodImpl(MethodImplOptions.AggressiveInlining)]
193     protected override TLinkAddress GetTreeRoot(TLinkAddress link) =>
194         ↪ GetLinkIndexPartReference(link).RootAsSource;
195
196     /// <summary>
197     /// <para>
198     /// Gets the base part value using the specified link.
199     /// </para>
200     /// <para></para>
201     /// </summary>
202     /// <param name="link">
203     /// <para>The link.</para>
204     /// <para></para>
205     /// </param>
206     /// <returns>
207     /// <para>The link</para>
208     /// <para></para>
209     /// </returns>
210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
211     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
212         ↪ GetLinkDataPartReference(link).Source;
213
214     /// <summary>
215     /// <para>
216     /// Gets the key part value using the specified link.
217     /// </para>
218     /// <para></para>
219     /// </summary>
220     /// <param name="link">
221     /// <para>The link.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>

```

```

225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 protected override TLinkAddress GetKeyPartValue(TLinkAddress link) =>
    ↪ 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) =>
    ↪ SearchCore(GetTreeRoot(source), target);
266 }
267 }

```

1.43 ./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) =>
    ↪ 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) =>
108         ↪ GetLinkIndexPartReference(node).RightAsSource;
109
110     /// <summary>
111     /// <para>
112     /// Sets the left using the specified node.
113     /// </para>
114     /// <para></para>
115     /// </summary>
116     /// <param name="node">
117     /// <para>The node.</para>
118     /// <para></para>
119     /// </param>
120     /// <param name="left">
121     /// <para>The left.</para>
122     /// <para></para>
123     /// </param>
124     [MethodImpl(MethodImplOptions.AggressiveInlining)]
125     protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
126         ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
127
128     /// <summary>
129     /// <para>
130     /// Sets the right using the specified node.
131     /// </para>
132     /// <para></para>
133     /// </summary>
134     /// <param name="node">
135     /// <para>The node.</para>
136     /// <para></para>
137     /// </param>
138     /// <param name="right">
139     /// <para>The right.</para>
140     /// <para></para>
141     /// </param>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
144         ↪ GetLinkIndexPartReference(node).RightAsSource = right;
145
146     /// <summary>
147     /// <para>
148     /// Gets the size using the specified node.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="node">
153     /// <para>The node.</para>
154     /// <para></para>
155     /// </param>
156     /// <returns>
157     /// <para>The link</para>
158     /// <para></para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     protected override TLinkAddress GetSize(TLinkAddress node) =>
162         ↪ GetLinkIndexPartReference(node).SizeAsSource;
163
164     /// <summary>
165     /// <para>
166     /// Sets the size using the specified node.
167     /// </para>
168     /// <para></para>
169     /// </summary>
170     /// <param name="node">
171     /// <para>The node.</para>
172     /// <para></para>
173     /// </param>
174     /// <param name="size">
175     /// <para>The size.</para>
176     /// <para></para>
177     /// </param>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
179     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
180         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;

```

```

176
177     /// <summary>
178     /// <para>
179     /// Gets the tree root using the specified link.
180     /// </para>
181     /// <para></para>
182     /// </summary>
183     /// <param name="link">
184     /// <para>The link.</para>
185     /// <para></para>
186     /// </param>
187     /// <returns>
188     /// <para>The link</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]
192     protected override TLinkAddress GetTreeRoot(TLinkAddress link) =>
193         ↪ GetLinkIndexPartReference(link).RootAsSource;
194
195     /// <summary>
196     /// <para>
197     /// Gets the base part value using the specified link.
198     /// </para>
199     /// <para></para>
200     /// </summary>
201     /// <param name="link">
202     /// <para>The link.</para>
203     /// <para></para>
204     /// </param>
205     /// <returns>
206     /// <para>The link</para>
207     /// <para></para>
208     /// </returns>
209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
210     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
211         ↪ GetLinkDataPartReference(link).Source;
212
213     /// <summary>
214     /// <para>
215     /// Gets the key part value using the specified link.
216     /// </para>
217     /// <para></para>
218     /// </summary>
219     /// <param name="link">
220     /// <para>The link.</para>
221     /// <para></para>
222     /// </param>
223     /// <returns>
224     /// <para>The link</para>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     protected override TLinkAddress GetKeyPartValue(TLinkAddress link) =>
229         ↪ GetLinkDataPartReference(link).Target;
230
231     /// <summary>
232     /// <para>
233     /// Clears the node using the specified node.
234     /// </para>
235     /// <para></para>
236     /// </summary>
237     /// <param name="node">
238     /// <para>The node.</para>
239     /// <para></para>
240     /// </param>
241     [MethodImpl(MethodImplOptions.AggressiveInlining)]
242     protected override void ClearNode(TLinkAddress node)
243     {
244         ref var link = ref GetLinkIndexPartReference(node);
245         link.LeftAsSource = Zero;
246         link.RightAsSource = Zero;
247         link.SizeAsSource = Zero;
248     }
249
250     /// <summary>
251     /// <para>
252     /// Searches the source.
253     /// </para>

```

```

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

```

1.44 ./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> :
        ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see
19         ↪ cref="InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddr_
41         ↪ ess> 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>

```

```

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) =>
    ↪ GetLinkIndexPartReference(node).RightAsTarget;
108
109 /// <summary>
110 /// <para>
111 /// Sets the left using the specified node.
112 /// </para>
113 /// <para></para>
114 /// </summary>
115 /// <param name="node">
116 /// <para>The node.</para>
117 /// <para></para>
118 /// </param>
119 /// <param name="left">
120 /// <para>The left.</para>
121 /// <para></para>
122 /// </param>
123 [MethodImpl(MethodImplOptions.AggressiveInlining)]
124 protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
    ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
125
126 /// <summary>

```



```

127     /// <para>
128     /// Sets the right using the specified node.
129     /// </para>
130     /// <para></para>
131     /// </summary>
132     /// <param name="node">
133     /// <para>The node.</para>
134     /// <para></para>
135     /// </param>
136     /// <param name="right">
137     /// <para>The right.</para>
138     /// <para></para>
139     /// </param>
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     protected override void SetRight(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 using the specified link.
181     /// </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     /// </summary>
200     /// <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).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/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
        ↳ 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) =>
91         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <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).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        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(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>

```

```

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    /// </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).SizeAsTarget = 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).RootAsTarget;
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).Target;
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).Source;

```

```

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.LeftAsTarget = Zero;
243         link.RightAsTarget = Zero;
244         link.SizeAsTarget = Zero;
245     }
246
247     /// <summary>
248     /// <para>
249     /// Searches the source.
250     /// </para>
251     /// <para></para>
252     /// </summary>
253     /// <param name="source">
254     /// <para>The source.</para>
255     /// <para></para>
256     /// </param>
257     /// <param name="target">
258     /// <para>The target.</para>
259     /// <para></para>
260     /// </param>
261     /// <returns>
262     /// <para>The link</para>
263     /// <para></para>
264     /// </returns>
265     public override TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
266         ↪ SearchCore(GetTreeRoot(target), source);
267 }

```

1.46 ./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
        ↪ FileMappedResizableDirectMemory(dataMemory), new
        ↪ FileMappedResizableDirectMemory(indexMemory)) { }
44
45     /// <summary>
46     /// <para>
47     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     /// <param name="dataMemory">
52     /// <para>A data memory.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="indexMemory">
56     /// <para>A index memory.</para>
57     /// <para></para>
58     /// </param>
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
61
62     /// <summary>
63     /// <para>
64     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="dataMemory">
69     /// <para>A data memory.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="indexMemory">
73     /// <para>A index memory.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="memoryReservationStep">
77     /// <para>A memory reservation step.</para>
78     /// <para></para>
79     /// </param>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↪ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
        ↪ IndexTreeType.Default, useLinkedList: true) { }
82
83     /// <summary>
84     /// <para>
85     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <param name="dataMemory">
90     /// <para>A data memory.</para>
91     /// <para></para>
92     /// </param>
93     /// <param name="indexMemory">
94     /// <para>A index memory.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="memoryReservationStep">
98     /// <para>A memory reservation step.</para>
99     /// <para></para>
100    /// </param>
101    /// <param name="constants">
102    /// <para>A constants.</para>
103    /// <para></para>
104    /// </param>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

106 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants) :
    ↪ this(dataMemory, indexMemory, memoryReservationStep, constants,
    ↪ IndexTreeType.Default, useLinkedList: true) { }
107
108 /// <summary>
109 /// <para>
110 /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
111 /// </para>
112 /// </summary>
113
114 /// <param name="dataMemory">
115 /// <para>A data memory.</para>
116 /// </param>
117
118 /// <param name="indexMemory">
119 /// <para>A index memory.</para>
120 /// </param>
121
122 /// <param name="memoryReservationStep">
123 /// <para>A memory reservation step.</para>
124 /// </param>
125
126 /// <param name="constants">
127 /// <para>A constants.</para>
128 /// </param>
129
130 /// <param name="indexTreeType">
131 /// <para>A index tree type.</para>
132 /// </param>
133
134 /// <param name="useLinkedList">
135 /// <para>A use linked list.</para>
136 /// </param>
137
138 [MethodImpl(MethodImplOptions.AggressiveInlining)]
139 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants,
    ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
    ↪ memoryReservationStep, constants, useLinkedList)
140 {
141     if (indexTreeType == IndexTreeType.SizeBalancedTree)
142     {
143         _createInternalSourceTreeMethods = () => new
            ↪ InternalLinksSourcesSizeBalancedTreeMethods<TLinkAddress>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
144         _createExternalSourceTreeMethods = () => new
            ↪ ExternalLinksSourcesSizeBalancedTreeMethods<TLinkAddress>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
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,
174    ↪ IResizableDirectMemory indexMemory)
175    {
176        _linksDataParts = (byte*)dataMemory.Pointer;
177        _linksIndexParts = (byte*)indexMemory.Pointer;
178        _header = _linksIndexParts;
179        if (_useLinkedList)
180        {
181            InternalSourcesListMethods = new
182            ↪ InternalLinksSourcesLinkedListMethods<TLinkAddress>(Constants,
183            ↪ _linksDataParts, _linksIndexParts);
184        }
185        else
186        {
187            InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
188        }
189        ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
190        InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
191        ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
192        UnusedLinksListMethods = new UnusedLinksListMethods<TLinkAddress>(_linksDataParts,
193        ↪ _header);
194    }
195
196    /// <summary>
197    /// <para>
198    /// Resets the pointers.
199    /// </para>
200    /// <para></para>
201    /// </summary>
202    [MethodImpl(MethodImplOptions.AggressiveInlining)]
203    protected override void ResetPointers()
204    {
205        base.ResetPointers();
206        _linksDataParts = null;
207        _linksIndexParts = null;
208        _header = null;
209    }
210
211    /// <summary>
212    /// <para>
213    /// Gets the header reference.
214    /// </para>
215    /// <para></para>
216    /// </summary>
217    /// <returns>
218    /// <para>A ref links header of t link</para>
219    /// <para></para>
220    /// </returns>
221    [MethodImpl(MethodImplOptions.AggressiveInlining)]
222    protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
223    ↪ AsRef<LinksHeader<TLinkAddress>>(_header);
224
225    /// <summary>
226    /// <para>
227    /// Gets the link data part reference using the specified link index.
228    /// </para>
229    /// <para></para>
230    /// </summary>
231    /// <param name="linkIndex">
232    /// <para>The link index.</para>
233    /// <para></para>
234    /// </param>
235    /// <returns>
236    /// <para>A ref raw link data part of t link</para>
237    /// <para></para>
238    /// </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 }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Disposables;
5 using Platform.Singletons;
6 using Platform.Converters;
7 using Platform.Numbers;
8 using Platform.Memory;
9 using Platform.Data.Exceptions;
10 using Platform.Delegates;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Memory.Split.Generic
15 {
16     /// <summary>
17     /// <para>
18     /// Represents the split memory links base.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <seealso cref="DisposableBase"/>
23     /// <seealso cref="ILinks{TLinkAddress}"/>
24     public abstract class SplitMemoryLinksBase<TLinkAddress> : DisposableBase,
        ↪ ILinks<TLinkAddress>
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     /// <summary>
146     /// Возвращает общее число связей находящихся в хранилище.
147     /// </summary>
148     protected virtual TLinkAddress Total
149     {
150         [MethodImpl(MethodImplOptions.AggressiveInlining)]
151         get
152         {
153             ref var header = ref GetHeaderReference();
154             return Subtract(header.AllocatedLinks, header.FreeLinks);
155         }
156     }
157     /// <summary>
158     /// <para>
159     /// Gets the constants value.
160     /// </para>
161     /// <para></para>
162     /// </summary>
163     public virtual LinksConstants<TLinkAddress> Constants
164     {
165         [MethodImpl(MethodImplOptions.AggressiveInlining)]
166         get;
167     }
168     /// <summary>
169     /// <para>
170     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
171     /// </para>
172     /// <para></para>
173     /// </summary>
174     /// <param name="dataMemory">
175     /// <para>A data memory.</para>
176     /// <para></para>
177     /// </param>
178     /// <param name="indexMemory">
179     /// <para>A index memory.</para>
180     /// <para></para>
181     /// </param>
182     /// <param name="memoryReservationStep">
183     /// <para>A memory reservation step.</para>
184     /// <para></para>
185     /// </param>
186     /// <param name="constants">
187     /// <para>A constants.</para>
188     /// <para></para>
189     /// </param>
190     /// <param name="useLinkedList">
191     /// <para>A use linked list.</para>
192     /// <para></para>
193     /// </param>
194     [MethodImpl(MethodImplOptions.AggressiveInlining)]
195     protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
196     indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants,
197     bool useLinkedList)

```

```

197 {
198     _dataMemory = dataMemory;
199     _indexMemory = indexMemory;
200     _dataMemoryReservationStepInBytes = memoryReservationStep * LinkDataPartSizeInBytes;
201     _indexMemoryReservationStepInBytes = memoryReservationStep *
        ↳ LinkIndexPartSizeInBytes;
202     _useLinkedList = useLinkedList;
203     Constants = constants;
204 }
205
206 /// <summary>
207 /// <para>
208 /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
209 /// </para>
210 /// <para></para>
211 /// </summary>
212 /// <param name="dataMemory">
213 /// <para>A data memory.</para>
214 /// <para></para>
215 /// </param>
216 /// <param name="indexMemory">
217 /// <para>A index memory.</para>
218 /// <para></para>
219 /// </param>
220 /// <param name="memoryReservationStep">
221 /// <para>A memory reservation step.</para>
222 /// <para></para>
223 /// </param>
224 [MethodImpl(MethodImplOptions.AggressiveInlining)]
225 protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↳ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
        ↳ useLinkedList: true) { }
226
227 /// <summary>
228 /// <para>
229 /// Inits the data memory.
230 /// </para>
231 /// <para></para>
232 /// </summary>
233 /// <param name="dataMemory">
234 /// <para>The data memory.</para>
235 /// <para></para>
236 /// </param>
237 /// <param name="indexMemory">
238 /// <para>The index memory.</para>
239 /// <para></para>
240 /// </param>
241 [MethodImpl(MethodImplOptions.AggressiveInlining)]
242 protected virtual void Init(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory)
243 {
244     // Read allocated links from header
245     if (indexMemory.ReservedCapacity < LinkHeaderSizeInBytes)
246     {
247         indexMemory.ReservedCapacity = LinkHeaderSizeInBytes;
248     }
249     SetPointers(dataMemory, indexMemory);
250     ref var header = ref GetHeaderReference();
251     var allocatedLinks = ConvertToInt64(header.AllocatedLinks);
252     // Adjust reserved capacity
253     var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
254     if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
255     {
256         minimumDataReservedCapacity = dataMemory.UsedCapacity;
257     }
258     if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
259     {
260         minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
261     }
262     var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
263     if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
264     {
265         minimumIndexReservedCapacity = indexMemory.UsedCapacity;
266     }
267     if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
268     {
269         minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
270     }
271 }

```

```

271 // Check for alignment
272 if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
273 {
274     minimumDataReservedCapacity = ((minimumDataReservedCapacity /
275     ↪ _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
276     ↪ _dataMemoryReservationStepInBytes;
277 }
278 if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)
279 {
280     minimumIndexReservedCapacity = ((minimumIndexReservedCapacity /
281     ↪ _indexMemoryReservationStepInBytes) * _indexMemoryReservationStepInBytes) +
282     ↪ _indexMemoryReservationStepInBytes;
283 }
284 if (dataMemory.ReservedCapacity != minimumDataReservedCapacity)
285 {
286     dataMemory.ReservedCapacity = minimumDataReservedCapacity;
287 }
288 if (indexMemory.ReservedCapacity != minimumIndexReservedCapacity)
289 {
290     indexMemory.ReservedCapacity = minimumIndexReservedCapacity;
291 }
292 SetPointers(dataMemory, indexMemory);
293 header = ref GetHeaderReference();
294 // Ensure correctness _memory.UsedCapacity over _header->AllocatedLinks
295 // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
296 dataMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
297     ↪ LinkDataPartSizeInBytes) + LinkDataPartSizeInBytes; // First link is read only
298     ↪ zero link.
299 indexMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
300     ↪ LinkIndexPartSizeInBytes) + LinkHeaderSizeInBytes;
301 // Ensure correctness _memory.ReservedLinks over _header->ReservedCapacity
302 // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
303 header.ReservedLinks = ConvertToAddress((dataMemory.ReservedCapacity -
304     ↪ LinkDataPartSizeInBytes) / LinkDataPartSizeInBytes);
305 }
306
307 /// <summary>
308 /// <para>
309 /// Counts the substitution.
310 /// </para>
311 /// <para></para>
312 /// </summary>
313 /// <param name="restriction">
314 /// <para>The substitution.</para>
315 /// <para></para>
316 /// </param>
317 /// <exception cref="NotSupportedException">
318 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
319 /// <para></para>
320 /// </exception>
321 /// <returns>
322 /// <para>The link</para>
323 /// <para></para>
324 /// </returns>
325 [MethodImpl(MethodImplOptions.AggressiveInlining)]
326 public virtual TLinkAddress Count(ICollection<TLinkAddress>? restriction)
327 {
328     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
329     if (restriction.Count == 0)
330     {
331         return Total;
332     }
333     var constants = Constants;
334     var any = constants.Any;
335     var index = this.GetIndex(restriction);
336     if (restriction.Count == 1)
337     {
338         if (AreEqual(index, any))
339         {
340             return Total;
341         }
342         return Exists(index) ? GetOne() : GetZero();
343     }
344     if (restriction.Count == 2)
345     {
346         var value = restriction[1];
347         if (AreEqual(index, any))
348         {
349

```

```

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 = this.GetSource(restriction);
389     var target = this.GetTarget(restriction);
390     if (AreEqual(index, any))
391     {
392         if (AreEqual(source, any) && AreEqual(target, any))
393         {
394             return Total;
395         }
396         else if (AreEqual(source, any))
397         {
398             if (externalReferencesRange.HasValue &&
399                 ↪ externalReferencesRange.Value.Contains(target))
400             {
401                 return ExternalTargetsTreeMethods.CountUsages(target);
402             }
403             else
404             {
405                 return InternalTargetsTreeMethods.CountUsages(target);
406             }
407         }
408         else if (AreEqual(target, any))
409         {
410             if (externalReferencesRange.HasValue &&
411                 ↪ externalReferencesRange.Value.Contains(source))
412             {
413                 return ExternalSourcesTreeMethods.CountUsages(source);
414             }
415             else
416             {
417                 if (_useLinkedList)
418                 {

```

```

412         return InternalSourcesListMethods.CountUsages(source);
413     }
414     else
415     {
416         return InternalSourcesTreeMethods.CountUsages(source);
417     }
418 }
419 }
420 else //if(source != Any && target != Any)
421 {
422     // ЭКВИВАЛЕНТ Exists(source, target) => Count(Any, source, target) > 0
423     TLinkAddress link;
424     if (externalReferencesRange.HasValue)
425     {
426         if (externalReferencesRange.Value.Contains(source) &&
427             ↪ externalReferencesRange.Value.Contains(target))
428         {
429             link = ExternalSourcesTreeMethods.Search(source, target);
430         }
431         else if (externalReferencesRange.Value.Contains(source))
432         {
433             link = InternalTargetsTreeMethods.Search(source, target);
434         }
435         else if (externalReferencesRange.Value.Contains(target))
436         {
437             if (_useLinkedList)
438             {
439                 link = ExternalSourcesTreeMethods.Search(source, target);
440             }
441             else
442             {
443                 link = InternalSourcesTreeMethods.Search(source, target);
444             }
445         }
446         else
447         {
448             if (_useLinkedList ||
449                 ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
450                 ↪ InternalTargetsTreeMethods.CountUsages(target)))
451             {
452                 link = InternalTargetsTreeMethods.Search(source, target);
453             }
454             else
455             {
456                 link = InternalSourcesTreeMethods.Search(source, target);
457             }
458         }
459     }
460     }
461     else
462     {
463         if (_useLinkedList ||
464             ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
465             ↪ InternalTargetsTreeMethods.CountUsages(target)))
466         {
467             link = InternalTargetsTreeMethods.Search(source, target);
468         }
469         else
470         {
471             link = InternalSourcesTreeMethods.Search(source, target);
472         }
473     }
474     return AreEqual(link, constants.Null) ? GetZero() : GetOne();
475 }
476 }
477 else
478 {
479     if (!Exists(index))
480     {
481         return GetZero();
482     }
483     if (AreEqual(source, any) && AreEqual(target, any))
484     {
485         return GetOne();
486     }
487     ref var storedLinkValue = ref GetLinkDataPartReference(index);
488     if (!AreEqual(source, any) && !AreEqual(target, any))
489     {

```



```

484         if (AreEqual(storedLinkValue.Source, source) &&
485             ⇨ AreEqual(storedLinkValue.Target, target))
486         {
487             return GetOne();
488         }
489         return GetZero();
490     }
491     var value = default(TLinkAddress);
492     if (AreEqual(source, any))
493     {
494         value = target;
495     }
496     if (AreEqual(target, any))
497     {
498         value = source;
499     }
500     if (AreEqual(storedLinkValue.Source, value) ||
501         ⇨ AreEqual(storedLinkValue.Target, value))
502     {
503         return GetOne();
504     }
505     return GetZero();
506 }
507 throw new NotSupportedException("Другие размеры и способы ограничений не
508 ⇨ поддерживаются.");
509 }
510
511 /// <summary>
512 /// <para>
513 /// Eaches the handler.
514 /// </para>
515 /// <para></para>
516 /// </summary>
517 /// <param name="handler">
518 /// <para>The handler.</para>
519 /// <para></para>
520 /// </param>
521 /// <param name="restriction">
522 /// <para>The substitution.</para>
523 /// <para></para>
524 /// </param>
525 /// <exception cref="NotSupportedException">
526 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
527 /// <para></para>
528 /// </exception>
529 /// <returns>
530 /// <para>The link</para>
531 /// <para></para>
532 /// </returns>
533 [MethodImpl(MethodImplOptions.AggressiveInlining)]
534 public virtual TLinkAddress Each(IList<TLinkAddress>? restriction,
535     ⇨ ReadHandler<TLinkAddress>? handler)
536 {
537     var constants = Constants;
538     var @break = constants.Break;
539     if (restriction.Count == 0)
540     {
541         for (var link = GetOne(); LessOrEqualThan(link,
542             ⇨ GetHeaderReference().AllocatedLinks); link = Increment(link))
543         {
544             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
545             {
546                 return @break;
547             }
548         }
549         return @break;
550     }
551     var @continue = constants.Continue;
552     var any = constants.Any;
553     var index = this.GetIndex(restriction);
554     if (restriction.Count == 1)
555     {
556         if (AreEqual(index, any))
557         {
558             return Each(Array.Empty<TLinkAddress>(), handler);
559         }
560         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),
572             ↪ @break))
573         {
574             return @break;
575         }
576         return Each(new Link<TLinkAddress>(index, any, value), handler);
577     }
578     else
579     {
580         if (!Exists(index))
581         {
582             return @continue;
583         }
584         if (AreEqual(value, any))
585         {
586             return handler(GetLinkStruct(index));
587         }
588         ref var storedLinkValue = ref GetLinkDataPartReference(index);
589         if (AreEqual(storedLinkValue.Source, value) ||
590             AreEqual(storedLinkValue.Target, value))
591         {
592             return handler(GetLinkStruct(index));
593         }
594         return @continue;
595     }
596 }
597 if (restriction.Count == 3)
598 {
599     var externalReferencesRange = constants.ExternalReferencesRange;
600     var source = this.GetSource(restriction);
601     var target = this.GetTarget(restriction);
602     if (AreEqual(index, any))
603     {
604         if (AreEqual(source, any) && AreEqual(target, any))
605         {
606             return Each(Array.Empty<TLinkAddress>(), handler);
607         }
608         else if (AreEqual(source, any))
609         {
610             if (externalReferencesRange.HasValue &&
611                 ↪ externalReferencesRange.Value.Contains(target))
612             {
613                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
614             }
615             else
616             {
617                 return InternalTargetsTreeMethods.EachUsage(target, handler);
618             }
619         }
620         else if (AreEqual(target, any))
621         {
622             if (externalReferencesRange.HasValue &&
623                 ↪ externalReferencesRange.Value.Contains(source))
624             {
625                 return ExternalSourcesTreeMethods.EachUsage(source, handler);
626             }
627             else
628             {
629                 if (_useLinkedList)
630                 {
631                     return InternalSourcesListMethods.EachUsage(source, handler);
632                 }
633             }
634         }
635     }
636 }

```

```

632         return InternalSourcesTreeMethods.EachUsage(source, handler);
633     }
634 }
635 }
636 else //if(source != Any && target != Any)
637 {
638     TLinkAddress link;
639     if (externalReferencesRange.HasValue)
640     {
641         if (externalReferencesRange.Value.Contains(source) &&
642             ↪ externalReferencesRange.Value.Contains(target))
643         {
644             link = ExternalSourcesTreeMethods.Search(source, target);
645         }
646         else if (externalReferencesRange.Value.Contains(source))
647         {
648             link = InternalTargetsTreeMethods.Search(source, target);
649         }
650         else if (externalReferencesRange.Value.Contains(target))
651         {
652             if (_useLinkedList)
653             {
654                 link = ExternalSourcesTreeMethods.Search(source, target);
655             }
656             else
657             {
658                 link = InternalSourcesTreeMethods.Search(source, target);
659             }
660         }
661         else
662         {
663             if (_useLinkedList ||
664                 ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
665                 ↪ InternalTargetsTreeMethods.CountUsages(target)))
666             {
667                 link = InternalTargetsTreeMethods.Search(source, target);
668             }
669             else
670             {
671                 link = InternalSourcesTreeMethods.Search(source, target);
672             }
673         }
674     }
675     else
676     {
677         if (_useLinkedList ||
678             ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
679             ↪ InternalTargetsTreeMethods.CountUsages(target)))
680         {
681             link = InternalTargetsTreeMethods.Search(source, target);
682         }
683         else
684         {
685             link = InternalSourcesTreeMethods.Search(source, target);
686         }
687     }
688     return AreEqual(link, constants.Null) ? @continue :
689         ↪ handler(GetLinkStruct(link));
690 }
691 }
692 else
693 {
694     if (!Exists(index))
695     {
696         return @continue;
697     }
698     if (AreEqual(source, any) && AreEqual(target, any))
699     {
700         return handler(GetLinkStruct(index));
701     }
702     ref var storedLinkValue = ref GetLinkDataPartReference(index);
703     if (!AreEqual(source, any) && !AreEqual(target, any))
704     {
705         if (AreEqual(storedLinkValue.Source, source) &&
706             AreEqual(storedLinkValue.Target, target))
707         {
708             return handler(GetLinkStruct(index));
709         }
710     }

```

```

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 = this.GetIndex(restriction);
736     var before = GetLinkStruct(linkIndex);
737     ref var link = ref GetLinkDataPartReference(linkIndex);
738     var source = link.Source;
739     var target = link.Target;
740     ref var header = ref GetHeaderReference();
741     ref var rootAsSource = ref header.RootAsSource;
742     ref var rootAsTarget = ref header.RootAsTarget;
743     // Будет корректно работать только в том случае, если пространство выделенной связи
    ↳ предварительно заполнено нулями
744     if (!AreEqual(source, @null))
745     {
746         if (externalReferencesRange.HasValue &&
    ↳ externalReferencesRange.Value.Contains(source))
747         {
748             ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
749         }
750         else
751         {
752             if (_useLinkedList)
753             {
754                 InternalSourcesListMethods.Detach(source, linkIndex);
755             }
756             else
757             {
758                 InternalSourcesTreeMethods.Detach(ref
    ↳ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
759             }
760         }
761     }
762     if (!AreEqual(target, @null))
763     {
764         if (externalReferencesRange.HasValue &&
    ↳ externalReferencesRange.Value.Contains(target))
765         {
766             ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
767         }
768         else
769         {
770             InternalTargetsTreeMethods.Detach(ref
    ↳ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
771         }
772     }
773     source = link.Source = this.GetSource(substitution);

```

```

774 target = link.Target = this.GetTarget(substitution);
775 if (!AreEqual(source, @null))
776 {
777     if (externalReferencesRange.HasValue &&
778         ↪ externalReferencesRange.Value.Contains(source))
779     {
780         ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
781     }
782     else
783     {
784         if (_useLinkedList)
785         {
786             InternalSourcesListMethods.AttachAsLast(source, linkIndex);
787         }
788         else
789         {
790             InternalSourcesTreeMethods.Attach(ref
791                 ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
792         }
793     }
794 }
795 if (!AreEqual(target, @null))
796 {
797     if (externalReferencesRange.HasValue &&
798         ↪ externalReferencesRange.Value.Contains(target))
799     {
800         ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
801     }
802     else
803     {
804         InternalTargetsTreeMethods.Attach(ref
805             ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
806     }
807 }
808 return handler != null ? handler(before, new Link<TLinkAddress>(linkIndex, source,
809     ↪ target)) : Constants.Continue;
810 }
811
812 /// <remarks>
813 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
814 ↪ пространство
815 /// </remarks>
816 [MethodImpl(MethodImplOptions.AggressiveInlining)]
817 public virtual TLinkAddress Create(IList<TLinkAddress>? substitution,
818     ↪ WriteHandler<TLinkAddress>? handler)
819 {
820     ref var header = ref GetHeaderReference();
821     var freeLink = header.FirstFreeLink;
822     if (!AreEqual(freeLink, Constants.Null))
823     {
824         UnusedLinksListMethods.Detach(freeLink);
825     }
826     else
827     {
828         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
829         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
830         {
831             throw new
832                 ↪ LinksLimitReachedException<TLinkAddress>(maximumPossibleInnerReference);
833         }
834         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
835         {
836             _dataMemory.ReservedCapacity += _dataMemory.ReservationStepInBytes;
837             _indexMemory.ReservedCapacity += _indexMemory.ReservationStepInBytes;
838             SetPointers(_dataMemory, _indexMemory);
839             header = ref GetHeaderReference();
840             header.ReservedLinks = ConvertToAddress(_dataMemory.ReservedCapacity /
841                 ↪ LinkDataPartSizeInBytes);
842         }
843         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
844         _dataMemory.UsedCapacity += LinkDataPartSizeInBytes;
845         _indexMemory.UsedCapacity += LinkIndexPartSizeInBytes;
846     }
847     return handler != null ? handler(null, GetLinkStruct(freeLink)) : Constants.Continue;
848 }
849
850 /// <summary>
851 /// <para>

```

```

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

```

```

916 [MethodImpl(MethodImplOptions.AggressiveInlining)]
917 protected virtual void ResetPointers()
918 {
919     InternalSourcesListMethods = null;
920     InternalSourcesTreeMethods = null;
921     ExternalSourcesTreeMethods = null;
922     InternalTargetsTreeMethods = null;
923     ExternalTargetsTreeMethods = null;
924     UnusedLinksListMethods = null;
925 }
926
927 /// <summary>
928 /// <para>
929 /// Gets the header reference.
930 /// </para>
931 /// <para></para>
932 /// </summary>
933 /// <returns>
934 /// <para>A ref links header of t link</para>
935 /// <para></para>
936 /// </returns>
937 [MethodImpl(MethodImplOptions.AggressiveInlining)]
938 protected abstract ref LinksHeader<TLinkAddress> GetHeaderReference();
939
940 /// <summary>
941 /// <para>
942 /// Gets the link data part reference using the specified link index.
943 /// </para>
944 /// <para></para>
945 /// </summary>
946 /// <param name="linkIndex">
947 /// <para>The link index.</para>
948 /// <para></para>
949 /// </param>
950 /// <returns>
951 /// <para>A ref raw link data part of t link</para>
952 /// <para></para>
953 /// </returns>
954 [MethodImpl(MethodImplOptions.AggressiveInlining)]
955 protected abstract ref RawLinkDataPart<TLinkAddress>
956     ↪ GetLinkDataPartReference(TLinkAddress linkIndex);
957
958 /// <summary>
959 /// <para>
960 /// Gets the link index part reference using the specified link index.
961 /// </para>
962 /// <para></para>
963 /// </summary>
964 /// <param name="linkIndex">
965 /// <para>The link index.</para>
966 /// <para></para>
967 /// </param>
968 /// <returns>
969 /// <para>A ref raw link index part of t link</para>
970 /// <para></para>
971 /// </returns>
972 [MethodImpl(MethodImplOptions.AggressiveInlining)]
973 protected abstract ref RawLinkIndexPart<TLinkAddress>
974     ↪ GetLinkIndexPartReference(TLinkAddress linkIndex);
975
976 /// <summary>
977 /// <para>
978 /// Determines whether this instance exists.
979 /// </para>
980 /// <para></para>
981 /// </summary>
982 /// <param name="link">
983 /// <para>The link.</para>
984 /// <para></para>
985 /// </param>
986 /// <returns>
987 /// <para>The bool</para>
988 /// <para></para>
989 /// </returns>
990 [MethodImpl(MethodImplOptions.AggressiveInlining)]
991 protected virtual bool Exists(TLinkAddress link)
992     => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
993     && 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>
1041     /// <para>
1042     /// Gets the zero.
1043     /// </para>
1044     /// <para></para>
1045     /// </summary>
1046     /// <returns>
1047     /// <para>The link</para>
1048     /// <para></para>
1049     /// </returns>
1050     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1051     protected virtual TLinkAddress GetZero() => default;
1052
1053     /// <summary>
1054     /// <para>
1055     /// Determines whether this instance are equal.
1056     /// </para>
1057     /// <para></para>
1058     /// </summary>
1059     /// <param name="first">
1060     /// <para>The first.</para>
1061     /// <para></para>
1062     /// </param>
1063     /// <param name="second">
1064     /// <para>The second.</para>
1065     /// <para></para>
1066     /// </param>
1067     /// <returns>
1068     /// <para>The bool</para>
1069     /// <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)]
1113     protected virtual bool LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
1114         ↪ _comparer.Compare(first, second) <= 0;
1115
1116     /// <summary>
1117     /// <para>
1118     /// Determines whether this instance greater than.
1119     /// </para>
1120     /// <para></para>
1121     /// </summary>
1122     /// <param name="first">
1123     /// <para>The first.</para>
1124     /// <para></para>
1125     /// </param>
1126     /// <param name="second">
1127     /// <para>The second.</para>
1128     /// <para></para>
1129     /// </param>
1130     /// <returns>
1131     /// <para>The bool</para>
1132     /// <para></para>
1133     /// </returns>
1134     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1135     protected virtual bool GreaterThan(TLinkAddress first, TLinkAddress second) =>
1136         ↪ _comparer.Compare(first, second) > 0;
1137
1138     /// <summary>
1139     /// <para>
1140     /// Determines whether this instance greater or equal than.
1141     /// </para>
1142     /// <para></para>
1143     /// </summary>
1144     /// <param name="first">

```

```

1141    /// <para>The first.</para>
1142    /// <para></para>
1143    /// </param>
1144    /// <param name="second">
1145    /// <para>The second.</para>
1146    /// <para></para>
1147    /// </param>
1148    /// <returns>
1149    /// <para>The bool</para>
1150    /// <para></para>
1151    /// </returns>
1152    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1153    protected virtual bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
        ↪ _comparer.Compare(first, second) >= 0;

1154
1155    /// <summary>
1156    /// <para>
1157    /// Converts the to int 64 using the specified value.
1158    /// </para>
1159    /// <para></para>
1160    /// </summary>
1161    /// <param name="value">
1162    /// <para>The value.</para>
1163    /// <para></para>
1164    /// </param>
1165    /// <returns>
1166    /// <para>The long</para>
1167    /// <para></para>
1168    /// </returns>
1169    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1170    protected virtual long ConvertToInt64(TLinkAddress value) =>
        ↪ _addressToInt64Converter.Convert(value);

1171
1172    /// <summary>
1173    /// <para>
1174    /// Converts the to address using the specified value.
1175    /// </para>
1176    /// <para></para>
1177    /// </summary>
1178    /// <param name="value">
1179    /// <para>The value.</para>
1180    /// <para></para>
1181    /// </param>
1182    /// <returns>
1183    /// <para>The link</para>
1184    /// <para></para>
1185    /// </returns>
1186    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1187    protected virtual TLinkAddress ConvertToAddress(long value) =>
        ↪ _int64ToAddressConverter.Convert(value);

1188
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) =>
        ↪ Arithmetic<TLinkAddress>.Add(first, second);

1209
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
1232     /// <summary>
1233     /// <para>
1234     /// Increments the link.
1235     /// </para>
1236     /// <para></para>
1237     /// </summary>
1238     /// <param name="link">
1239     /// <para>The link.</para>
1240     /// <para></para>
1241     /// </param>
1242     /// <returns>
1243     /// <para>The link</para>
1244     /// <para></para>
1245     /// </returns>
1246     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1247     protected virtual TLinkAddress Increment(TLinkAddress link) =>
1248         ↪ Arithmetic<TLinkAddress>.Increment(link);
1249
1250     /// <summary>
1251     /// <para>
1252     /// Decrements the link.
1253     /// </para>
1254     /// <para></para>
1255     /// </summary>
1256     /// <param name="link">
1257     /// <para>The link.</para>
1258     /// <para></para>
1259     /// </param>
1260     /// <returns>
1261     /// <para>The link</para>
1262     /// <para></para>
1263     /// </returns>
1264     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1265     protected virtual TLinkAddress Decrement(TLinkAddress link) =>
1266         ↪ Arithmetic<TLinkAddress>.Decrement(link);
1267
1268     #region Disposable
1269
1270     /// <summary>
1271     /// <para>
1272     /// Gets the allow multiple dispose calls value.
1273     /// </para>
1274     /// <para></para>
1275     /// </summary>
1276     protected override bool AllowMultipleDisposeCalls
1277     {
1278         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1279         get => true;
1280     }
1281
1282     /// <summary>
1283     /// <para>
1284     /// Disposes the manual.
1285     /// </para>
1286     /// <para></para>
1287     /// </summary>
1288     /// <param name="manual">
1289     /// <para>The manual.</para>
1290     /// <para></para>
1291     /// </param>
1292     /// <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 }

```

1.48 ./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> :
19     ↪ AbsoluteCircularDoublyLinkedListMethods<TLinkAddress>, ILinksListMethods<TLinkAddress>
20     {
21         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
22         ↪ = UncheckedConverter<TLinkAddress, long>.Default;
23         private readonly byte* _links;
24         private readonly byte* _header;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="links">
33         /// <para>A links.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UnusedLinksListMethods(byte* links, byte* header)
42         {
43             _links = links;
44             _header = header;
45         }
46
47         /// <summary>
48         /// <para>
49         /// Gets the header reference.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <returns>
54         /// <para>A ref links header of t link</para>
55         /// <para></para>
56         /// </returns>
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         protected virtual ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
59         ↪ AsRef<LinksHeader<TLinkAddress>>(_header);
60     }
61 }

```

```

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

```

```

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

```

```

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

```

1.49 ./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 =
            ↪ 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<RawLinkDataPart<TLinkAddress>>.Size;
27
28         /// <summary>
29         /// <para>
30         /// The source.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         public TLinkAddress Source;
35
36         /// <summary>
37         /// <para>
38         /// The target.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         public TLinkAddress Target;
43
44         /// <summary>
45         /// <para>
46         /// Determines whether this instance equals.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="obj">
51         /// <para>The obj.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>The bool</para>
56         /// <para></para>
57         /// </returns>
58         [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 }

```

1.50 ./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 =
            ↪ 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<RawLinkIndexPart<TLinkAddress>>.Size;
27
28         /// <summary>
29         /// <para>
30         /// The root as source.
31         /// </para>
32         /// <para></para>

```



```

33     /// </summary>
34     public TLinkAddress RootAsSource;
35     /// <summary>
36     /// <para>
37     /// The left as source.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public TLinkAddress LeftAsSource;
42     /// <summary>
43     /// <para>
44     /// The right as source.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     public TLinkAddress RightAsSource;
49     /// <summary>
50     /// <para>
51     /// The size as source.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     public TLinkAddress SizeAsSource;
56     /// <summary>
57     /// <para>
58     /// The root as target.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLinkAddress RootAsTarget;
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 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,
139        ↪ SizeAsSource, RootAsTarget, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
140
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    public static bool operator ==(RawLinkIndexPart<TLinkAddress> left,
143        ↪ RawLinkIndexPart<TLinkAddress> right) => left.Equals(right);
144
145    [MethodImpl(MethodImplOptions.AggressiveInlining)]
146    public static bool operator !=(RawLinkIndexPart<TLinkAddress> left,
147        ↪ RawLinkIndexPart<TLinkAddress> right) => !(left == right);
148
149    }
150
151    }

```

1.51 ./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    }
36 }

```

```

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="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="constants">
49     /// <para>A constants.</para>
50     /// <para></para>
51     /// </param>
52     /// <param name="linksDataParts">
53     /// <para>A links data parts.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="linksIndexParts">
57     /// <para>A links index parts.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="header">
61     /// <para>A header.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
66     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
67     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
68     {
69         LinksDataParts = linksDataParts;
70         LinksIndexParts = linksIndexParts;
71         Header = header;
72     }
73
74     /// <summary>
75     /// <para>
76     /// Gets the zero.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <returns>
81     /// <para>The link</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override TLinkAddress GetZero() => 0U;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance equal to zero.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="value">
94     /// <para>The value.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The bool</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override bool EqualToZero(TLinkAddress value) => value == 0U;
103
104    /// <summary>
105    /// <para>
106    /// Determines whether this instance are equal.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="first">
111    /// <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) =>
350         ↪ first - second;
351
352     /// <summary>
353     /// <para>
354     /// Gets the header reference.
355     /// </para>
356     /// <para></para>
357     /// </summary>
358     /// <returns>
359     /// <para>A ref links header of t link</para>
360     /// <para></para>
361     /// </returns>
362     [MethodImpl(MethodImplOptions.AggressiveInlining)]
363     protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref *Header;
364
365     /// <summary>
366     /// <para>
367     /// Gets the link data part reference using the specified link.
368     /// </para>
369     /// <para></para>
370     /// </summary>
371     /// <param name="link">
372     /// <para>The link.</para>
373     /// <para></para>
374     /// </param>
375     /// <returns>
376     /// <para>A ref raw link data part of t link</para>
377     /// <para></para>
378     /// </returns>
379     [MethodImpl(MethodImplOptions.AggressiveInlining)]
380     protected override ref RawLinkDataPart<TLinkAddress>
381     ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
382
383     /// <summary>
384     /// <para>
385     /// Gets the link index part reference using the specified link.
386     /// </para>
387     /// <para></para>
388     /// </summary>
389     /// <param name="link">
390     /// <para>The link.</para>
391     /// <para></para>
392     /// </param>
393     /// <returns>
394     /// <para>A ref raw link index part of t link</para>
395     /// <para></para>
396     /// </returns>
397     [MethodImpl(MethodImplOptions.AggressiveInlining)]
398     protected override ref RawLinkIndexPart<TLinkAddress>
399     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
400
401     /// <summary>
402     /// <para>
403     /// Determines whether this instance first is to the left of second.
404     /// </para>
405     /// <para></para>
406     /// </summary>
407     /// <param name="first">
408     /// <para>The first.</para>
409     /// <para></para>
410     /// </param>
411     /// <param name="second">
412     /// <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     }
}

```

1.52 ./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>

```



```

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="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
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 UInt32ExternalLinksSizeBalancedTreeMethodsBase(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 link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLinkAddress GetZero() => 0U;
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(TLinkAddress value) => value == 0U;
104
105    /// <summary>
106    /// <para>
107    /// Determines whether this instance are equal.
108    /// </para>
109    /// <para></para>
110    /// </summary>

```

```

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

```

```

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     }
}

```

1.53 ./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">

```

```

28     /// <para>A links data parts.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<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].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>
173    /// <para></para>
174    /// </param>
175    /// <param name="size">
176    /// <para>The size.</para>
177    /// <para></para>
178    /// </param>

```



```

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.
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 FirstIsToLeftOfSecond(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>

```

```

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     /// <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 }

```

1.54 ./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 :
        ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32ExternalLinksSourcesSizeBalancedTreeMethods"/>
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>

```

```

33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt32 ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<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     /// </summary>
116     /// <param name="node">
117     /// <para>The node.</para>
118     /// </para>
119     /// </param>
120     /// <param name="left">
121     /// <para>The left.</para>
122     /// </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     /// </summary>
133     /// <param name="node">
134     /// <para>The node.</para>
135     /// </para>
136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// </para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
143         ↪ LinksIndexParts[node].RightAsSource = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// </para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// </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     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// </para>
170     /// </param>
171     /// <param name="size">
172     /// <para>The size.</para>
173     /// </para>
174     /// </param>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
177         ↪ LinksIndexParts[node].SizeAsSource = size;

```

```

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

```

```

252     /// <param name="secondSource">
253     /// <para>The second source.</para>
254     /// </para>
255     /// </param>
256     /// <param name="secondTarget">
257     /// <para>The second target.</para>
258     /// </para>
259     /// </param>
260     /// <returns>
261     /// <para>The bool</para>
262     /// </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     /// <summary>
268     /// <para>
269     /// Clears the node using the specified node.
270     /// </para>
271     /// </para>
272     /// </summary>
273     /// <param name="node">
274     /// <para>The node.</para>
275     /// </para>
276     /// </param>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override void ClearNode(TLinkAddress node)
279     {
280         ref var link = ref LinksIndexParts[node];
281         link.LeftAsSource = Zero;
282         link.RightAsSource = Zero;
283         link.SizeAsSource = Zero;
284     }
285 }
286 }
287 }

```

1.55 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods

```

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>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
        ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
        ↪ cref="UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// </para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// </para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// </para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// </para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>

```

```

37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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">
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)]
180     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
181         ↪ LinksIndexParts[node].SizeAsTarget = size;
182
183     /// <summary>
184     /// <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->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 FirstIsToTheLeftOfSecond(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 FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
269        ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
270        => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
271        ↳ secondSource;

```

```

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/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) =>
    ↳ 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>

```

```

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 FirstIsToTheLeftOfSecond(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 FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
269     ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
270     => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
271     ↳ secondSource;

```

```

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 }

```

1.57 ./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
        ↪ cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
43         /// </para>
44         /// </para></para>
45         /// </summary>

```

```

46    /// <param name="constants">
47    /// <para>A constants.</para>
48    /// <para></para>
49    /// </param>
50    /// <param name="linksDataParts">
51    /// <para>A links data parts.</para>
52    /// <para></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 UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase(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 link</para>
79    /// <para></para>
80    /// </returns>
81    [MethodImpl(MethodImplOptions.AggressiveInlining)]
82    protected override TLinkAddress GetZero() => OU;
83
84    /// <summary>
85    /// <para>
86    /// Determines whether this instance equal to zero.
87    /// </para>
88    /// <para></para>
89    /// </summary>
90    /// <param name="value">
91    /// <para>The value.</para>
92    /// <para></para>
93    /// </param>
94    /// <returns>
95    /// <para>The bool</para>
96    /// <para></para>
97    /// </returns>
98    [MethodImpl(MethodImplOptions.AggressiveInlining)]
99    protected override bool EqualToZero(TLinkAddress value) => value == OU;
100
101    /// <summary>
102    /// <para>
103    /// Determines whether this instance are equal.
104    /// </para>
105    /// <para></para>
106    /// </summary>
107    /// <param name="first">
108    /// <para>The first.</para>
109    /// <para></para>
110    /// </param>
111    /// <param name="second">
112    /// <para>The second.</para>
113    /// <para></para>
114    /// </param>
115    /// <returns>
116    /// <para>The bool</para>
117    /// <para></para>
118    /// </returns>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override bool AreEqual(TLinkAddress first, TLinkAddress second) => first ==
    ↪ second;

```

```

121     /// <summary>
122     /// <para>
123     /// Determines whether this instance greater than zero.
124     /// </para>
125     /// <para></para>
126     /// </summary>
127     /// <param name="value">
128     /// <para>The value.</para>
129     /// <para></para>
130     /// </param>
131     /// <returns>
132     /// <para>The bool</para>
133     /// <para></para>
134     /// </returns>
135     [MethodImpl(MethodImplOptions.AggressiveInlining)]
136     protected override bool GreaterThanZero(TLinkAddress value) => value > 0U;
137
138     /// <summary>
139     /// <para>
140     /// Determines whether this instance greater than.
141     /// </para>
142     /// <para></para>
143     /// </summary>
144     /// <param name="first">
145     /// <para>The first.</para>
146     /// <para></para>
147     /// </param>
148     /// <param name="second">
149     /// <para>The second.</para>
150     /// <para></para>
151     /// </param>
152     /// <returns>
153     /// <para>The bool</para>
154     /// <para></para>
155     /// </returns>
156     [MethodImpl(MethodImplOptions.AggressiveInlining)]
157     protected override bool GreaterThan(TLinkAddress first, TLinkAddress second) => first >
158     ↪ 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)]

```



```

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     /// <summary>
274     /// <para>
275     /// Increments the value.
276     /// </para>
277     /// <para></para>
278     /// </summary>
279     /// <param name="value">
280     /// <para>The value.</para>
281     /// <para></para>
282     /// </param>
283     /// <returns>
284     /// <para>The link</para>
285     /// <para></para>
286     /// </returns>
287     [MethodImpl(MethodImplOptions.AggressiveInlining)]
288     protected override TLinkAddress Increment(TLinkAddress value) => ++value;

289     /// <summary>
290     /// <para>
291     /// Decrements the value.
292     /// </para>
293     /// <para></para>
294     /// </summary>
295     /// <param name="value">
296     /// <para>The value.</para>
297     /// <para></para>
298     /// </param>
299     /// <returns>
300     /// <para>The link</para>
301     /// <para></para>
302     /// </returns>
303     [MethodImpl(MethodImplOptions.AggressiveInlining)]
304     protected override TLinkAddress Decrement(TLinkAddress value) => --value;

305     /// <summary>
306     /// <para>
307     /// Adds the first.
308     /// </para>
309     /// <para></para>
310     /// </summary>
311     /// <param name="first">
312     /// <para>The first.</para>
313     /// <para></para>
314     /// </param>
315     /// <param name="second">
316     /// <para>The second.</para>
317     /// <para></para>
318     /// </param>
319     /// <returns>
320     /// <para>The link</para>
321     /// <para></para>
322     /// </returns>
323     [MethodImpl(MethodImplOptions.AggressiveInlining)]
324     protected override TLinkAddress Add(TLinkAddress first, TLinkAddress second) => first +
        ↳ second;

325     /// <summary>
326     /// <para>
327     /// Subtracts the first.
328     /// </para>
329     /// <para></para>
330     /// </summary>
331     /// <param name="first">
332     /// <para>The first.</para>
333     /// <para></para>
334     /// </param>
335     /// <param name="second">
336     /// <para>The second.</para>
337     /// <para></para>
338     /// </param>
339     /// <returns>
340     /// <para>The link</para>
341     /// <para></para>
342     /// </returns>
343     [MethodImpl(MethodImplOptions.AggressiveInlining)]
344     protected override TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) => first -
        ↳ second;

```

```

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    /// </summary>
356    /// <param name="link">
357    /// <para>The link.</para>
358    /// </para>
359    /// </param>
360    /// <returns>
361    /// <para>A ref raw link data part of t link</para>
362    /// </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    /// </summary>
373    /// <param name="link">
374    /// <para>The link.</para>
375    /// </para>
376    /// </param>
377    /// <returns>
378    /// <para>A ref raw link index part of t link</para>
379    /// </para>
380    /// </returns>
381    [MethodImpl(MethodImplOptions.AggressiveInlining)]
382    protected override ref RawLinkIndexPart<TLinkAddress>
383        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
384
385    /// <summary>
386    /// <para>
387    /// Determines whether this instance first is to the left of second.
388    /// </para>
389    /// </summary>
390    /// <param name="first">
391    /// <para>The first.</para>
392    /// </para>
393    /// </param>
394    /// <param name="second">
395    /// <para>The second.</para>
396    /// </para>
397    /// </param>
398    /// <returns>
399    /// <para>The bool</para>
400    /// </para>
401    /// </returns>
402    [MethodImpl(MethodImplOptions.AggressiveInlining)]
403    protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress
404        ↪ second) => GetKeyPartValue(first) < GetKeyPartValue(second);
405
406    /// <summary>
407    /// <para>
408    /// Determines whether this instance first is to the right of second.
409    /// </para>
410    /// </summary>
411    /// <param name="first">
412    /// <para>The first.</para>
413    /// </para>
414    /// </param>
415    /// <param name="second">
416    /// <para>The second.</para>
417    /// </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/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"/>
43         ↪ instance.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         /// <param name="constants">
48         /// <para>A constants.</para>
49         /// <para></para>
50         /// </param>
51         /// <param name="linksDataParts">
52         /// <para>A links data parts.</para>
53         /// <para></para>
54         /// </param>
55         /// <param name="linksIndexParts">
56         /// <para>A links index parts.</para>
57         /// <para></para>
58         /// </param>
59         /// <param name="header">
60         /// <para>A header.</para>
61         /// <para></para>
62         /// </param>
63         [MethodImpl(MethodImplOptions.AggressiveInlining)]
64         protected UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
65         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
66     {

```

```

66     LinksDataParts = linksDataParts;
67     LinksIndexParts = linksIndexParts;
68     Header = header;
69 }
70
71 /// <summary>
72 /// <para>
73 /// Gets the zero.
74 /// </para>
75 /// <para></para>
76 /// </summary>
77 /// <returns>
78 /// <para>The link</para>
79 /// <para></para>
80 /// </returns>
81 [MethodImpl(MethodImplOptions.AggressiveInlining)]
82 protected override 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    /// <summary>
160    /// <para>
161    /// <para>Determines whether this instance greater or equal than.
162    /// </para>
163    /// <para></para>
164    /// </summary>
165    /// <param name="first">
166    /// <para>The first.</para>
167    /// <para></para>
168    /// </param>
169    /// <param name="second">
170    /// <para>The second.</para>
171    /// <para></para>
172    /// </param>
173    /// <returns>
174    /// <para>The bool</para>
175    /// <para></para>
176    /// </returns>
177    [MethodImpl(MethodImplOptions.AggressiveInlining)]
178    protected override bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
179    ↪ first >= second;

180    /// <summary>
181    /// <para>
182    /// <para>Determines whether this instance greater or equal than zero.
183    /// </para>
184    /// <para></para>
185    /// </summary>
186    /// <param name="value">
187    /// <para>The value.</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 GreaterOrEqualThanZero(TLinkAddress value) => true; // value >=
196    ↪ 0 is always true for ulong

197    /// <summary>
198    /// <para>
199    /// <para>Determines whether this instance less or equal than zero.
200    /// </para>
201    /// <para></para>
202    /// </summary>
203    /// <param name="value">
204    /// <para>The value.</para>
205    /// <para></para>
206    /// </param>
207    /// <returns>
208    /// <para>The bool</para>
209    /// <para></para>
210    /// </returns>
211    [MethodImpl(MethodImplOptions.AggressiveInlining)]
212    protected override bool LessOrEqualThanZero(TLinkAddress value) => value == OUL; //
213    ↪ value is always >= 0 for ulong

214    /// <summary>
215    /// <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) =>
    ↪ 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 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 }

```

1.59 ./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      }
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  }

```

1.60 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalance

```

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 :
16        ↳ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see
21        ↳ cref="UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
43        ↳ nkAddress> 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        /// <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>
153    /// </summary>
154    /// <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>
222    /// <param name="node">
223    /// <para>The node.</para>
224    /// <para></para>
225    /// </param>
226    /// <returns>
227    /// <para>The link</para>
228    /// <para></para>
229    /// </returns>

```

```

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/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>

```

```

29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt32InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<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>
173    /// <para></para>
174    /// </param>
175    /// <param name="size">
176    /// <para>The size.</para>
177    /// <para></para>
178    /// </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) =>
    ↳ LinksDataParts[node].Source;

211
212 /// <summary>
213 /// <para>
214 /// Gets the key part value using the specified node.
215 /// </para>
216 /// <para></para>
217 /// </summary>
218 /// <param name="node">
219 /// <para>The node.</para>
220 /// <para></para>
221 /// </param>
222 /// <returns>
223 /// <para>The link</para>
224 /// <para></para>
225 /// </returns>
226 [MethodImpl(MethodImplOptions.AggressiveInlining)]
227 protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
    ↳ LinksDataParts[node].Target;

228
229 /// <summary>
230 /// <para>
231 /// Clears the node using the specified node.
232 /// </para>
233 /// <para></para>
234 /// </summary>
235 /// <param name="node">
236 /// <para>The node.</para>
237 /// <para></para>
238 /// </param>
239 [MethodImpl(MethodImplOptions.AggressiveInlining)]
240 protected override void ClearNode(TLinkAddress node)
241 {
242     ref var link = ref LinksIndexParts[node];
243     link.LeftAsSource = Zero;
244     link.RightAsSource = Zero;
245     link.SizeAsSource = Zero;
246 }
247
248 /// <summary>

```

```

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

```

1.62 ./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 :
        ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
        ↪ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 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
58         ↳ LinksIndexParts[node].LeftAsTarget;
59
60     /// <summary>
61     /// <para>
62     /// Gets the right reference using the specified node.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="node">
67     /// <para>The node.</para>
68     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The ref link</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
76         ↳ LinksIndexParts[node].RightAsTarget;
77
78     /// <summary>
79     /// <para>
80     /// Gets the left using the specified node.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="node">
85     /// <para>The node.</para>
86     /// <para></para>
87     /// </param>
88     /// <returns>
89     /// <para>The link</para>
90     /// <para></para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     protected override TLinkAddress GetLeft(TLinkAddress node) =>
94         ↳ LinksIndexParts[node].LeftAsTarget;
95
96     /// <summary>
97     /// <para>
98     /// Gets the right using the specified node.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    /// <param name="node">
103    /// <para>The node.</para>
104    /// <para></para>
105    /// </param>
106    /// <returns>
107    /// <para>The link</para>
108    /// <para></para>
109    /// </returns>
110    [MethodImpl(MethodImplOptions.AggressiveInlining)]
111    protected override TLinkAddress GetRight(TLinkAddress node) =>
112        ↳ LinksIndexParts[node].RightAsTarget;
113
114    /// <summary>
115    /// <para>
116    /// Sets the left using the specified node.
117    /// </para>
118    /// <para></para>
119    /// </summary>
120    /// <param name="node">
121    /// <para>The node.</para>
122    /// <para></para>
123    /// </param>
124    /// <param name="left">
125    /// <para>The left.</para>
126    /// <para></para>
127    /// </param>
128    [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     /// </summary>
133     /// <param name="node">
134     /// <para>The node.</para>
135     /// </param>
136     /// <param name="right">
137     /// <para>The right.</para>
138     /// </param>
139     [MethodImpl(MethodImplOptions.AggressiveInlining)]
140     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
141         ↳ LinksIndexParts[node].RightAsTarget = right;
142
143     /// <summary>
144     /// <para>
145     /// Gets the size using the specified node.
146     /// </para>
147     /// </summary>
148     /// <param name="node">
149     /// <para>The node.</para>
150     /// </param>
151     /// <returns>
152     /// <para>The link</para>
153     /// </returns>
154     [MethodImpl(MethodImplOptions.AggressiveInlining)]
155     protected override TLinkAddress GetSize(TLinkAddress node) =>
156         ↳ LinksIndexParts[node].SizeAsTarget;
157
158     /// <summary>
159     /// <para>
160     /// Sets the size using the specified node.
161     /// </para>
162     /// </summary>
163     /// <param name="node">
164     /// <para>The node.</para>
165     /// </param>
166     /// <param name="size">
167     /// <para>The size.</para>
168     /// </param>
169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
170     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
171         ↳ LinksIndexParts[node].SizeAsTarget = size;
172
173     /// <summary>
174     /// <para>
175     /// Gets the tree root using the specified node.
176     /// </para>
177     /// </summary>
178     /// <param name="node">
179     /// <para>The node.</para>
180     /// </param>
181     /// <returns>
182     /// <para>The link</para>
183     /// </returns>
184     [MethodImpl(MethodImplOptions.AggressiveInlining)]
185     protected override TLinkAddress GetTreeRoot(TLinkAddress node) =>
186         ↳ LinksIndexParts[node].RootAsTarget;
187
188     /// <summary>
189     /// <para>
190

```

```

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].Target;
212
213     /// <summary>
214     /// <para>
215     /// Gets the key part value using the specified node.
216     /// </para>
217     /// <para></para>
218     /// </summary>
219     /// <param name="node">
220     /// <para>The node.</para>
221     /// <para></para>
222     /// </param>
223     /// <returns>
224     /// <para>The link</para>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
229         ↪ LinksDataParts[node].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 LinksIndexParts[node];
245         link.LeftAsTarget = Zero;
246         link.RightAsTarget = Zero;
247         link.SizeAsTarget = Zero;
248     }
249
250     /// <summary>
251     /// <para>
252     /// Searches the source.
253     /// </para>
254     /// <para></para>
255     /// </summary>
256     /// <param name="source">
257     /// <para>The source.</para>
258     /// <para></para>
259     /// </param>
260     /// <param name="target">
261     /// <para>The target.</para>
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(target), source);

```

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

```

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 :
16         ↳ UInt32InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32InternalLinksTargetsSizeBalancedTreeMethods"/>
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 UInt32InternalLinksTargetsSizeBalancedTreeMethods(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].LeftAsTarget;
64
65         /// <summary>
66         /// <para>
67         /// Gets the right reference using the specified node.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         /// <param name="node">
72         /// <para>The node.</para>
73         /// <para></para>
74         /// </param>
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].RightAsTarget;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override 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    /// </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    /// </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].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    /// </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].RightAsTarget = right;

```

```

144    /// <summary>
145    /// <para>
146    /// Gets the size using the specified node.
147    /// </para>
148    /// <para></para>
149    /// </summary>
150    /// <param name="node">
151    /// <para>The node.</para>
152    /// <para></para>
153    /// </param>
154    /// <returns>
155    /// <para>The link</para>
156    /// <para></para>
157    /// </returns>
158    [MethodImpl(MethodImplOptions.AggressiveInlining)]
159    protected override 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 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].RootAsTarget;
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].Target;
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].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 }

```

1.64 ./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     }
26 }

```

```

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 /// </summary>
34
35 /// <param name="dataMemory">
36 /// <para>A data memory.</para>
37 /// </param>
38
39 /// <param name="indexMemory">
40 /// <para>A index memory.</para>
41 /// </param>
42
43 [MethodImpl(MethodImplOptions.AggressiveInlining)]
44 public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46 /// <summary>
47 /// <para>
48 /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
49 /// </para>
50 /// </summary>
51
52 /// <param name="dataMemory">
53 /// <para>A data memory.</para>
54 /// </param>
55
56 /// <param name="indexMemory">
57 /// <para>A index memory.</para>
58 /// </param>
59
60 /// <param name="memoryReservationStep">
61 /// <para>A memory reservation step.</para>
62 /// </param>
63
64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 public UInt32SplitMemoryLinks(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="UInt32SplitMemoryLinks"/> instance.
70 /// </para>
71 /// </summary>
72
73 /// <param name="dataMemory">
74 /// <para>A data memory.</para>
75 /// </param>
76
77 /// <param name="indexMemory">
78 /// <para>A index memory.</para>
79 /// </param>
80
81 /// <param name="memoryReservationStep">
82 /// <para>A memory reservation step.</para>
83 /// </param>
84
85 /// <param name="constants">
86 /// <para>A constants.</para>
87 /// </param>
88
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
        ↳ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
135            _createExternalSourceTreeMethods = () => new
        ↳ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
136            _createInternalTargetTreeMethods = () => new
        ↳ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
137            _createExternalTargetTreeMethods = () => new
        ↳ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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,
158   ↳ IResizableDirectMemory indexMemory)
159 {
160     _linksDataParts = (RawLinkDataPart<TLinkAddress>*)dataMemory.Pointer;
161     _linksIndexParts = (RawLinkIndexPart<TLinkAddress>*)indexMemory.Pointer;
162     _header = (LinksHeader<TLinkAddress>*)indexMemory.Pointer;
163     if (_useLinkedList)
164     {
165         InternalSourcesListMethods = new
166             ↳ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
167             ↳ _linksIndexParts);
168     }
169     else
170     {
171         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
172         ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
173         InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
174         ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
175         UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
176     }
177 }
178 /// <summary>
179 /// <para>
180 /// Resets the pointers.
181 /// </para>
182 /// </summary>
183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
184 protected override void ResetPointers()
185 {
186     base.ResetPointers();
187     _linksDataParts = null;
188     _linksIndexParts = null;
189     _header = null;
190 }
191 /// <summary>
192 /// <para>
193 /// Gets the header reference.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <returns>
198 /// <para>A ref links header of t link</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override ref LinksHeader<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">

```

```

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

237
238    /// <summary>
239    /// <para>
240    /// Determines whether this instance are equal.
241    /// </para>
242    /// <para></para>
243    /// </summary>
244    /// <param name="first">
245    /// <para>The first.</para>
246    /// <para></para>
247    /// </param>
248    /// <param name="second">
249    /// <para>The second.</para>
250    /// <para></para>
251    /// </param>
252    /// <returns>
253    /// <para>The bool</para>
254    /// <para></para>
255    /// </returns>
256    [MethodImpl(MethodImplOptions.AggressiveInlining)]
257    protected override bool AreEqual(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress 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(TLinkAddress first, TLinkAddress second) =>
        ↪ first >= second;
342
343    /// <summary>
344    /// <para>
345    /// Gets the zero.
346    /// </para>
347    /// <para></para>
348    /// </summary>
349    /// <returns>
350    /// <para>The link</para>
351    /// <para></para>
352    /// </returns>
353    [MethodImpl(MethodImplOptions.AggressiveInlining)]
354    protected override 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 +
423         ↪ second;
424
425     /// <summary>
426     /// <para>
427     /// Subtracts the first.
428     /// </para>
429     /// <para></para>
430     /// </summary>
431     /// <param name="first">
432     /// <para>The first.</para>
433     /// <para></para>
434     /// </param>
435     /// <param name="second">
436     /// <para>The second.</para>
437     /// <para></para>
438     /// </param>
439     /// <returns>
440     /// <para>The link</para>
441     /// <para></para>
442     /// </returns>
443     [MethodImpl(MethodImplOptions.AggressiveInlining)]
444     protected override TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
445         ↪ first - second;
446
447     /// <summary>
448     /// <para>
449     /// Increments the link.
450     /// </para>
451     /// <para></para>
452     /// </summary>
453     /// <param name="link">
454     /// <para>The link.</para>
455     /// <para></para>

```

```

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

```

1.65 ./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,
37             ↪ LinksHeader<TLinkAddress>* 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>

```



```

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

```

1.66 ./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 :
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
29         /// <summary>
30         /// <para>
31         /// The links index parts.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
36
37         /// <summary>
38         /// <para>
39         /// The header.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         protected new readonly LinksHeader<TLinkAddress>* Header;
44
45         /// <summary>
46         /// <para>
47         /// Initializes a new <see
48         ↪ cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <param name="constants">
53         /// <para>A constants.</para>
54         /// <para></para>
55         /// </param>

```

```

51     /// <param name="linksDataParts">
52     /// <para>A links data parts.</para>
53     /// </para>
54     /// </param>
55     /// <param name="linksIndexParts">
56     /// <para>A links index parts.</para>
57     /// </para>
58     /// </param>
59     /// <param name="header">
60     /// <para>A header.</para>
61     /// </para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected UInt64 ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLi
    ↪ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
65         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
66     {
67         LinksDataParts = linksDataParts;
68         LinksIndexParts = linksIndexParts;
69         Header = header;
70     }
71
72     /// <summary>
73     /// <para>
74     /// Gets the zero.
75     /// </para>
76     /// </para>
77     /// </summary>
78     /// <returns>
79     /// <para>The ulong</para>
80     /// </para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override ulong GetZero() => OUL;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equal to zero.
88     /// </para>
89     /// </para>
90     /// </summary>
91     /// <param name="value">
92     /// <para>The value.</para>
93     /// </para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// </para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    protected override bool EqualToZero(ulong value) => value == OUL;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance are equal.
105    /// </para>
106    /// </para>
107    /// </summary>
108    /// <param name="first">
109    /// <para>The first.</para>
110    /// </para>
111    /// </param>
112    /// <param name="second">
113    /// <para>The second.</para>
114    /// </para>
115    /// </param>
116    /// <returns>
117    /// <para>The bool</para>
118    /// </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
198     ↪ always true for ulong
199
200     /// <summary>
201     /// <para>
202     /// Determines whether this instance less or equal than zero.
203     /// </para>
204     /// <para></para>

```

```

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">
411 /// <para>The second.</para>
412 /// <para></para>
413 /// </param>
414 /// <returns>
415 /// <para>The bool</para>
416 /// <para></para>
417 /// </returns>
418 [MethodImpl(MethodImplOptions.AggressiveInlining)]
419 protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
420 {
421     ref var firstLink = ref LinksDataParts[first];
422     ref var secondLink = ref LinksDataParts[second];
423     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
424     ↪ secondLink.Source, secondLink.Target);
425 }
426
427 /// <summary>
428 /// <para>
429 /// Determines whether this instance first is to the right of second.
430 /// </para>
431 /// <para></para>
432 /// </summary>
433 /// <param name="first">
434 /// <para>The first.</para>
435 /// <para></para>

```

```

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 }

```

1.67 ./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 :
        ↪ ExternalLinksSizeBalancedTreeMethodsBase<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 cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
        ↪ instance.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         /// <param name="constants">
48         /// <para>A constants.</para>
49         /// <para></para>
50         /// </param>
51         /// <param name="linksDataParts">
52         /// <para>A links data parts.</para>
53         /// <para></para>
54         /// </param>

```

```

55     /// <param name="linksIndexParts">
56     /// <para>A links index parts.</para>
57     /// </para>
58     /// </param>
59     /// <param name="header">
60     /// <para>A header.</para>
61     /// </para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected UInt64 ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65     {
66     }
67     LinksDataParts = linksDataParts;
68     LinksIndexParts = linksIndexParts;
69     Header = header;
70 }
71
72     /// <summary>
73     /// <para>
74     /// Gets the zero.
75     /// </para>
76     /// </para>
77     /// </summary>
78     /// <returns>
79     /// <para>The ulong</para>
80     /// </para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override ulong GetZero() => 0UL;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equal to zero.
88     /// </para>
89     /// </para>
90     /// </summary>
91     /// <param name="value">
92     /// <para>The value.</para>
93     /// </para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// </para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    protected override bool EqualToZero(ulong value) => value == 0UL;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance are equal.
105    /// </para>
106    /// </para>
107    /// </summary>
108    /// <param name="first">
109    /// <para>The first.</para>
110    /// </para>
111    /// </param>
112    /// <param name="second">
113    /// <para>The second.</para>
114    /// </para>
115    /// </param>
116    /// <returns>
117    /// <para>The bool</para>
118    /// </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>
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
198     ↪ always true for ulong
199
200     /// <summary>
201     /// <para>
202     /// Determines whether this instance less or equal than zero.
203     /// </para>
204     /// <para></para>
205     /// </summary>
206     /// <param name="value">
207     /// <para>The value.</para>
208     /// <para></para>
209     /// </param>
210     /// <returns>
211     /// <para>The bool</para>
212     /// <para></para>
213     /// </returns>
214     [MethodImpl(MethodImplOptions.AggressiveInlining)]
215     protected override bool LessOrEqualThanZero(ulong value) => false; // value < 0 is
216     ↪ always false for ulong

```

```

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

```

```

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

```

```

362     protected override ref LinkHeader<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">
411     /// <para>The second.</para>
412     /// <para></para>
413     /// </param>
414     /// <returns>
415     /// <para>The bool</para>
416     /// <para></para>
417     /// </returns>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
420     {
421         ref var firstLink = ref LinksDataParts[first];
422         ref var secondLink = ref LinksDataParts[second];
423         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
424             ↪ secondLink.Source, secondLink.Target);
425     }
426
427     /// <summary>
428     /// <para>
429     /// Determines whether this instance first is to the right of second.
430     /// </para>
431     /// <para></para>
432     /// </summary>
433     /// <param name="first">
434     /// <para>The first.</para>
435     /// <para></para>
436     /// </param>
437     /// <param name="second">
438     /// <para>The second.</para>
439     /// <para></para>
440     /// </param>
441     /// <returns>
442     /// <para>The bool</para>
443     /// <para></para>
444     /// </returns>

```

```

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 }

```

1.68 ./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 :
    ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
    ↪ cref="UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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) =>
143        ↪ LinksIndexParts[node].RightAsSource = right;
144
145    /// <summary>
146    /// <para>
147    /// Gets the size using the specified node.
148    /// </para>
149    /// <para></para>
150    /// </summary>
151    /// <param name="node">
152    /// <para>The node.</para>
153    /// <para></para>
154    /// </param>
155    /// <returns>
156    /// <para>The link</para>
157    /// <para></para>
158    /// </returns>
159    [MethodImpl(MethodImplOptions.AggressiveInlining)]
160    protected override TLinkAddress GetSize(TLinkAddress node) =>
161        ↪ LinksIndexParts[node].SizeAsSource;
162
163    /// <summary>
164    /// <para>
165    /// Sets the size using the specified node.
166    /// </para>
167    /// <para></para>
168    /// </summary>
169    /// <param name="node">
170    /// <para>The node.</para>
171    /// <para></para>
172    /// </param>
173    /// <param name="size">
174    /// <para>The size.</para>
175    /// <para></para>
176    /// </param>
177    [MethodImpl(MethodImplOptions.AggressiveInlining)]
178    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
179        ↪ LinksIndexParts[node].SizeAsSource = size;
180
181    /// <summary>
182    /// <para>
183    /// Gets the tree root.
184    /// </para>
185    /// <para></para>
186    /// </summary>
187    /// <returns>
188    /// <para>The link</para>
189    /// <para></para>
190    /// </returns>
191    [MethodImpl(MethodImplOptions.AggressiveInlining)]
192    protected override TLinkAddress GetTreeRoot() => Header->RootAsSource;
193
194    /// <summary>
195    /// <para>
196    /// Gets the base part value using the specified node.
197    /// </para>
198    /// <para></para>
199    /// </summary>
200    /// <param name="node">
201    /// <para>The node.</para>
202    /// <para></para>
203    /// </param>
204    /// <returns>
205    /// <para>The link</para>

```

```

203 /// <para></para>
204 /// </returns>
205 [MethodImpl(MethodImplOptions.AggressiveInlining)]
206 protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
    ↳ LinksDataParts[node].Source;
207
208 /// <summary>
209 /// <para>
210 /// Determines whether this instance first is to the left of second.
211 /// </para>
212 /// <para></para>
213 /// </summary>
214 /// <param name="firstSource">
215 /// <para>The first source.</para>
216 /// <para></para>
217 /// </param>
218 /// <param name="firstTarget">
219 /// <para>The first target.</para>
220 /// <para></para>
221 /// </param>
222 /// <param name="secondSource">
223 /// <para>The second source.</para>
224 /// <para></para>
225 /// </param>
226 /// <param name="secondTarget">
227 /// <para>The second target.</para>
228 /// <para></para>
229 /// </param>
230 /// <returns>
231 /// <para>The bool</para>
232 /// <para></para>
233 /// </returns>
234 [MethodImpl(MethodImplOptions.AggressiveInlining)]
235 protected override bool FirstIsToLeftOfSecond(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 }

```

1.69 ./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 :
16     ↪ UInt64ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt64ExternalLinksSourcesSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt64ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<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;

```

```

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     ↪ LinksIndexParts[node].RightAsSource;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected override TLinkAddress GetLeft(TLinkAddress node) =>
93     ↪ LinksIndexParts[node].LeftAsSource;
94
95     /// <summary>
96     /// <para>
97     /// Gets the right using the specified node.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="node">
102    /// <para>The node.</para>
103    /// <para></para>
104    /// </param>
105    /// <returns>
106    /// <para>The link</para>
107    /// <para></para>
108    /// </returns>
109    [MethodImpl(MethodImplOptions.AggressiveInlining)]
110    protected override TLinkAddress GetRight(TLinkAddress node) =>
111    ↪ LinksIndexParts[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    ↪ LinksIndexParts[node].LeftAsSource = left;
130
131    /// <summary>
132    /// <para>
133    /// Sets the right using the specified node.
134    /// </para>
135    /// <para></para>
136    /// </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) =>
143         ↪ LinksIndexParts[node].RightAsSource = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <param name="node">
152     /// <para>The node.</para>
153     /// <para></para>
154     /// </param>
155     /// <returns>
156     /// <para>The link</para>
157     /// <para></para>
158     /// </returns>
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     protected override TLinkAddress GetSize(TLinkAddress node) =>
161         ↪ LinksIndexParts[node].SizeAsSource;
162
163     /// <summary>
164     /// <para>
165     /// Sets the size using the specified node.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="node">
170     /// <para>The node.</para>
171     /// <para></para>
172     /// </param>
173     /// <param name="size">
174     /// <para>The size.</para>
175     /// <para></para>
176     /// </param>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
179         ↪ LinksIndexParts[node].SizeAsSource = size;
180
181     /// <summary>
182     /// <para>
183     /// Gets the tree root.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <returns>
188     /// <para>The link</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]
192     protected override TLinkAddress GetTreeRoot() => Header->RootAsSource;
193
194     /// <summary>
195     /// <para>
196     /// Gets the base part value using the specified node.
197     /// </para>
198     /// <para></para>
199     /// </summary>
200     /// <param name="node">
201     /// <para>The node.</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 node) =>
210         ↪ LinksDataParts[node].Source;

```

```

207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance first is to the left of second.
211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <param name="firstSource">
215     /// <para>The first source.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(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 }

```

1.70 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods

```

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 :
16     ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
43         ↪ nkAddress> 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].LeftAsTarget;
64
65         /// <summary>
66         /// <para>
67         /// Gets the right reference using the specified node.
68         /// </para>
69         /// <para></para>
70         /// </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) =>
        ↪ LinksIndexParts[node].LeftAsTarget = left;

126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="node">
134    /// <para>The node.</para>
135    /// <para></para>
136    /// </param>
137    /// <param name="right">
138    /// <para>The right.</para>

```

```

139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(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     /// </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].SizeAsTarget;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// </para>
170     /// </param>
171     /// <param name="size">
172     /// <para>The size.</para>
173     /// </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.
182     /// </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->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     /// </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 node) =>
206         ↳ LinksDataParts[node].Target;
207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance first is to the left of second.
211     /// </para>
212     /// <para></para>

```

```

213     /// </summary>
214     /// <param name="firstSource">
215     /// <para>The first source.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
236         => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
            ↪ secondSource;
237
238     /// <summary>
239     /// <para>
240     /// <para>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     /// <para>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 }

```


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

```

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 :
16         ↳ UInt64ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt64ExternalLinksTargetsSizeBalancedTreeMethods"/>
21         ↳ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt64ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<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].LeftAsTarget;
64
65         /// <summary>
66         /// <para>
67         /// Gets the right reference using the specified node.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         /// <param name="node">
72         /// <para>The node.</para>
73         /// <para></para>
74         /// </param>
75         /// <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         ↳ LinksIndexParts[node].RightAsTarget;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// </param>
85     /// </returns>
86     /// <para>The link</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     protected override TLinkAddress GetLeft(TLinkAddress node) =>
91         ↳ LinksIndexParts[node].LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// </param>
102    /// </returns>
103    /// <para>The link</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLinkAddress GetRight(TLinkAddress node) =>
108        ↳ 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    /// </param>
119    /// <param name="left">
120    /// <para>The left.</para>
121    /// </param>
122    /// </returns>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
125        ↳ LinksIndexParts[node].LeftAsTarget = left;
126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="node">
134    /// <para>The node.</para>
135    /// </param>
136    /// <param name="right">
137    /// <para>The right.</para>
138    /// </param>
139    /// </returns>
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
142        ↳ LinksIndexParts[node].RightAsTarget = right;

```

```

143
144     /// <summary>
145     /// <para>
146     /// Gets the size using the specified node.
147     /// </para>
148     /// <para></para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override 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>

```

```

218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
    ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
236     => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
    ↪ secondSource;

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     => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
    ↪ secondSource;

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

```

1.72 ./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 :
17        ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<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="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> 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 UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
66        ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
67        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
68        {
69            LinksDataParts = linksDataParts;
70            LinksIndexParts = linksIndexParts;
71            Header = header;
72        }
73
74        /// <summary>
75        /// <para>
76        /// Gets the zero.
77        /// </para>
78        /// <para></para>
79        /// </summary>
80        /// </returns>

```

```

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></para>
207    /// </param>
208    /// <returns>
209    /// <para>The bool</para>
210    /// <para></para>
211    /// </returns>
212    [MethodImpl(MethodImplOptions.AggressiveInlining)]
213    protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
214
215    /// <summary>
216    /// <para>
217    /// Determines whether this instance less or equal than.
218    /// </para>
219    /// <para></para>
220    /// </summary>
221    /// <param name="first">
222    /// <para>The first.</para>
223    /// <para></para>
224    /// </param>
225    /// <param name="second">
226    /// <para>The second.</para>
227    /// <para></para>
228    /// </param>
229    /// <returns>
230    /// <para>The bool</para>
231    /// <para></para>

```

```

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

```



```

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     /// <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/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 :
    ↪ 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="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
43     ↪ 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 UInt64InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
61     ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
62     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
63     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64     {
65         LinksDataParts = linksDataParts;
66         LinksIndexParts = linksIndexParts;
67         Header = header;
68     }
69
70     /// <summary>
71     /// <para>
72     /// Gets the zero.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <returns>
77     /// <para>The ulong</para>
78     /// <para></para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     protected override ulong GetZero() => 0UL;
82
83     /// <summary>
84     /// <para>
85     /// Determines whether this instance equal to zero.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <param name="value">
90     /// <para>The value.</para>
91     /// </param>
92     /// <returns>
93     /// <para>The bool</para>
94     /// <para></para>
95     /// </returns>
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     protected override bool EqualToZero(ulong value) => value == 0UL;
98
99     /// <summary>
100    /// <para>
101    /// Determines whether this instance are equal.
102    /// </para>
103    /// <para></para>
104    /// </summary>

```

```

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

```

```

185     /// <para></para>
186     /// </summary>
187     /// <param name="value">
188     /// <para>The value.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The bool</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
    ↪ always true for ulong

197
198     /// <summary>
199     /// <para>
200     /// Determines whether this instance less or equal than zero.
201     /// </para>
202     /// <para></para>
203     /// </summary>
204     /// <param name="value">
205     /// <para>The value.</para>
206     /// <para></para>
207     /// </param>
208     /// <returns>
209     /// <para>The bool</para>
210     /// <para></para>
211     /// </returns>
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]
213     protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↪ always >= 0 for ulong

214
215     /// <summary>
216     /// <para>
217     /// Determines whether this instance less or equal than.
218     /// </para>
219     /// <para></para>
220     /// </summary>
221     /// <param name="first">
222     /// <para>The first.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="second">
226     /// <para>The second.</para>
227     /// <para></para>
228     /// </param>
229     /// <returns>
230     /// <para>The bool</para>
231     /// <para></para>
232     /// </returns>
233     [MethodImpl(MethodImplOptions.AggressiveInlining)]
234     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
235
236     /// <summary>
237     /// <para>
238     /// Determines whether this instance less than zero.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="value">
243     /// <para>The value.</para>
244     /// <para></para>
245     /// </param>
246     /// <returns>
247     /// <para>The bool</para>
248     /// <para></para>
249     /// </returns>
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪ for ulong

252
253     /// <summary>
254     /// <para>
255     /// Determines whether this instance less than.
256     /// </para>
257     /// <para></para>
258     /// </summary>
259     /// <param name="first">

```

```

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

```

```

338     /// </param>
339     /// <param name="second">
340     /// <para>The second.</para>
341     /// <para></para>
342     /// </param>
343     /// <returns>
344     /// <para>The ulong</para>
345     /// <para></para>
346     /// </returns>
347     [MethodImpl(MethodImplOptions.AggressiveInlining)]
348     protected override ulong Subtract(ulong first, ulong second) => first - second;
349
350     /// <summary>
351     /// <para>
352     /// Gets the link data part reference using the specified link.
353     /// </para>
354     /// <para></para>
355     /// </summary>
356     /// <param name="link">
357     /// <para>The link.</para>
358     /// <para></para>
359     /// </param>
360     /// <returns>
361     /// <para>A ref raw link data part of t link</para>
362     /// <para></para>
363     /// </returns>
364     [MethodImpl(MethodImplOptions.AggressiveInlining)]
365     protected override ref RawLinkDataPart<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>

```

```

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 }

```

1.74 ./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)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
40        {
41            _linksDataParts = linksDataParts;
42            _linksIndexParts = linksIndexParts;
43        }
44
45        /// <summary>
46        /// <para>
47        /// Gets the link data part reference using the specified link.
48        /// </para>
49        /// <para></para>
50        /// </summary>
51        /// <param name="link">
52        /// <para>The link.</para>
53        /// <para></para>
54        /// </param>
55        /// <returns>
56        /// <para>A ref raw link data part of t link</para>
57        /// <para></para>
58        /// </returns>
59

```



```

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 }

```

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

```

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) =>
    ↪ 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     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// </param>
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
142         ↳ 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     /// </param>
153     /// <returns>
154     /// <para>The link</para>
155     /// </returns>
156     [MethodImpl(MethodImplOptions.AggressiveInlining)]
157     protected override TLinkAddress GetSize(TLinkAddress node) =>
158         ↳ LinksIndexParts[node].SizeAsSource;
159
160     /// <summary>
161     /// <para>
162     /// Sets the size using the specified node.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     /// <param name="node">
167     /// <para>The node.</para>
168     /// </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         ↳ LinksIndexParts[node].SizeAsSource = size;
175
176     /// <summary>
177     /// <para>
178     /// Gets the tree root using the specified node.
179     /// </para>
180     /// <para></para>
181     /// </summary>
182     /// <param name="node">
183     /// <para>The node.</para>
184     /// </param>
185     /// <returns>
186     /// <para>The link</para>
187     /// </returns>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override TLinkAddress GetTreeRoot(TLinkAddress node) =>
190         ↳ LinksIndexParts[node].RootAsSource;
191
192
193
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 }

```

1.76 ./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 :
16    ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see cref="UInt64InternalLinksSourcesSizeBalancedTreeMethods"/>
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 UInt64InternalLinksSourcesSizeBalancedTreeMethods(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);

```

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

```

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>

```



```

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 UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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 ulong</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref ulong GetLeftReference(ulong node) => ref
        ↳ LinksIndexParts[node].LeftAsTarget;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref ulong</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref ulong GetRightReference(ulong node) => ref
        ↳ LinksIndexParts[node].RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override 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 }

```

1.78 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMetho

```

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 :
        ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
20        ↪ instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="linksDataParts">
29        /// <para>A links data parts.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="linksIndexParts">
33        /// <para>A links index parts.</para>
34        /// <para></para>
35        /// </param>
36        /// <param name="header">
37        /// <para>A header.</para>
38        /// <para></para>
39        /// </param>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]
41        public UInt64InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<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 ulong</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref ulong GetLeftReference(ulong node) => ref
58         ↳ LinksIndexParts[node].LeftAsTarget;
59
60     /// <summary>
61     /// <para>
62     /// Gets the right reference using the specified node.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="node">
67     /// <para>The node.</para>
68     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The ref ulong</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected override ref ulong GetRightReference(ulong node) => ref
76         ↳ LinksIndexParts[node].RightAsTarget;
77
78     /// <summary>
79     /// <para>
80     /// Gets the left using the specified node.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="node">
85     /// <para>The node.</para>
86     /// <para></para>
87     /// </param>
88     /// <returns>
89     /// <para>The link</para>
90     /// <para></para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     protected override TLinkAddress GetLeft(TLinkAddress node) =>
94         ↳ LinksIndexParts[node].LeftAsTarget;
95
96     /// <summary>
97     /// <para>
98     /// Gets the right using the specified node.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    /// <param name="node">
103    /// <para>The node.</para>
104    /// <para></para>
105    /// </param>
106    /// <returns>
107    /// <para>The link</para>
108    /// <para></para>
109    /// </returns>
110    [MethodImpl(MethodImplOptions.AggressiveInlining)]
111    protected override TLinkAddress GetRight(TLinkAddress node) =>
112        ↳ LinksIndexParts[node].RightAsTarget;
113
114    /// <summary>
115    /// <para>
116    /// Sets the left using the specified node.
117    /// </para>
118    /// <para></para>
119    /// </summary>
120    /// <param name="node">
121    /// <para>The node.</para>
122    /// <para></para>
123    /// </param>
124    /// <returns>
125    /// <para>The link</para>
126    /// <para></para>
127    /// </returns>
128    [MethodImpl(MethodImplOptions.AggressiveInlining)]
129    protected override TLinkAddress GetLeft(TLinkAddress node) =>
130        ↳ LinksIndexParts[node].LeftAsTarget;
131
132    /// <summary>
133    /// <para>
134    /// Sets the right using the specified node.
135    /// </para>
136    /// <para></para>
137    /// </summary>
138    /// <param name="node">
139    /// <para>The node.</para>
140    /// <para></para>
141    /// </param>
142    /// <returns>
143    /// <para>The link</para>
144    /// <para></para>
145    /// </returns>
146    [MethodImpl(MethodImplOptions.AggressiveInlining)]
147    protected override TLinkAddress GetRight(TLinkAddress node) =>
148        ↳ LinksIndexParts[node].RightAsTarget;

```

```

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 }
```

1.79 ./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
45             ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
46
47         /// <summary>
48         /// <para>
49         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="dataMemory">
54         /// <para>A data memory.</para>
55         /// <para></para>
56         /// </param>
57         /// <param name="indexMemory">
58         /// <para>A index memory.</para>
59         /// <para></para>
60         /// </param>
61         /// <param name="memoryReservationStep">
62         /// <para>A memory reservation step.</para>
63         /// <para></para>
64         /// </param>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
67             ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
68             ↪ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
69             ↪ IndexTreeType.Default, useLinkedList: true) { }
70
71         /// <summary>
72         /// <para>
73         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
74         /// </para>
75         /// <para></para>
76         /// </summary>
77     }
```



```

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
135         ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
136         ↪ _linksDataParts, _linksIndexParts, _header);
137         _createExternalSourceTreeMethods = () => new
138         ↪ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
139         ↪ _linksDataParts, _linksIndexParts, _header);
140         _createInternalTargetTreeMethods = () => new
141         ↪ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
142         ↪ _linksDataParts, _linksIndexParts, _header);
143         _createExternalTargetTreeMethods = () => new
144         ↪ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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         ↪ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
175         ↪ _linksIndexParts);
176     }
177     else
178     {
179         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
180     }
181     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
182     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
183     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
184     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_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(ulong first, ulong second) => first == second;
260
261 /// <summary>
262 /// <para>
263 /// Determines whether this instance less than.
264 /// </para>
265 /// <para></para>
266 /// </summary>
267 /// <param name="first">
268 /// <para>The first.</para>
269 /// <para></para>
270 /// </param>
271 /// <param name="second">
272 /// <para>The second.</para>
273 /// <para></para>
274 /// </param>
275 /// <returns>
276 /// <para>The bool</para>
277 /// <para></para>
278 /// </returns>

```

```

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     /// <summary>
356     /// <para>
357     /// Gets the one.
358     /// </para>
359     /// <para></para>
360     /// </summary>
361     /// <returns>
362     /// <para>The ulong</para>
363     /// <para></para>
364     /// </returns>
365     [MethodImpl(MethodImplOptions.AggressiveInlining)]
366     protected override ulong GetOne() => 1UL;
367
368     /// <summary>
369     /// <para>
370     /// Converts the to int 64 using the specified value.
371     /// </para>
372     /// <para></para>
373     /// </summary>
374     /// <param name="value">
375     /// <para>The value.</para>
376     /// <para></para>
377     /// </param>
378     /// <returns>
379     /// <para>The long</para>
380     /// <para></para>
381     /// </returns>
382     [MethodImpl(MethodImplOptions.AggressiveInlining)]
383     protected override long ConvertToInt64(ulong value) => (long)value;
384
385     /// <summary>
386     /// <para>
387     /// Converts the to address using the specified value.
388     /// </para>
389     /// <para></para>
390     /// </summary>
391     /// <param name="value">
392     /// <para>The value.</para>
393     /// <para></para>
394     /// </param>
395     /// <returns>
396     /// <para>The ulong</para>
397     /// <para></para>
398     /// </returns>
399     [MethodImpl(MethodImplOptions.AggressiveInlining)]
400     protected override ulong ConvertToAddress(long value) => (ulong)value;
401
402     /// <summary>
403     /// <para>
404     /// Adds the first.
405     /// </para>
406     /// <para></para>
407     /// </summary>
408     /// <param name="first">
409     /// <para>The first.</para>
410     /// <para></para>
411     /// </param>
412     /// <param name="second">
413     /// <para>The second.</para>
414     /// <para></para>
415     /// </param>
416     /// <returns>
417     /// <para>The ulong</para>
418     /// <para></para>
419     /// </returns>
420     [MethodImpl(MethodImplOptions.AggressiveInlining)]
421     protected override ulong Add(ulong first, ulong second) => first + second;
422
423     /// <summary>
424     /// <para>
425     /// Subtracts the first.
426     /// </para>
427     /// <para></para>
428     /// </summary>
429     /// <param name="first">
430     /// <para>The first.</para>
431     /// <para></para>
432     /// </param>

```

```

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 }

```

1.80 ./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>*
    ↪ header)
37     : base((byte*)links, (byte*)header)
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 }

```

1.81 ./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> :
    ↪ SizedAndThreadedAVLBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
24     {
25         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
    ↪ = UncheckedConverter<TLinkAddress, long>.Default;
26         private static readonly UncheckedConverter<TLinkAddress, int> _addressToInt32Converter =
    ↪ UncheckedConverter<TLinkAddress, int>.Default;
27         private static readonly UncheckedConverter<bool, TLinkAddress> _boolToAddressConverter =
    ↪ UncheckedConverter<bool, TLinkAddress>.Default;
28         private static readonly UncheckedConverter<TLinkAddress, bool> _addressToBoolConverter =
    ↪ UncheckedConverter<TLinkAddress, bool>.Default;

```

```

29 private static readonly UncheckedConverter<int, TLinkAddress> _int32ToAddressConverter =
    ↳ UncheckedConverter<int, TLinkAddress>.Default;
30
31 /// <summary>
32 /// <para>
33 /// The break.
34 /// </para>
35 /// <para></para>
36 /// </summary>
37 protected readonly TLinkAddress Break;
38 /// <summary>
39 /// <para>
40 /// The continue.
41 /// </para>
42 /// <para></para>
43 /// </summary>
44 protected readonly TLinkAddress Continue;
45 /// <summary>
46 /// <para>
47 /// The links.
48 /// </para>
49 /// <para></para>
50 /// </summary>
51 protected readonly byte* Links;
52 /// <summary>
53 /// <para>
54 /// The header.
55 /// </para>
56 /// <para></para>
57 /// </summary>
58 protected readonly byte* Header;
59
60 /// <summary>
61 /// <para>
62 /// Initializes a new <see cref="LinksAvlBalancedTreeMethodsBase"/> instance.
63 /// </para>
64 /// <para></para>
65 /// </summary>
66 /// <param name="constants">
67 /// <para>A constants.</para>
68 /// <para></para>
69 /// </param>
70 /// <param name="links">
71 /// <para>A links.</para>
72 /// <para></para>
73 /// </param>
74 /// <param name="header">
75 /// <para>A header.</para>
76 /// <para></para>
77 /// </param>
78 [MethodImpl(MethodImplOptions.AggressiveInlining)]
79 protected LinksAvlBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, byte*
    ↳ links, byte* header)
80 {
81     Links = links;
82     Header = header;
83     Break = constants.Break;
84     Continue = constants.Continue;
85 }
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     /// <para>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     /// <para>Determines whether this instance first is to the left of second.
150     /// </para>
151     /// <para></para>
152     /// </summary>
153     /// <param name="source">
154     /// <para>The source.</para>
155     /// <para></para>
156     /// </param>
157     /// <param name="target">
158     /// <para>The target.</para>
159     /// <para></para>
160     /// </param>
161     /// <param name="rootSource">
162     /// <para>The root source.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="rootTarget">
166     /// <para>The root target.</para>
167     /// <para></para>
168     /// </param>
169     /// <returns>
170     /// <para>The bool</para>
171     /// <para></para>
172     /// </returns>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected abstract bool FirstIsToTheLeftOfSecond(TLinkAddress source, TLinkAddress
175     ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);
176
177     /// <summary>
178     /// <para>
179     /// <para>Gets the header reference.
180     /// </para>
181     /// <para></para>
182     /// </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
187        ↳ AsRef<LinksHeader<TLinkAddress>>(Header);
188
189    /// <summary>
190    /// <para>
191    /// Gets the link reference 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>A ref raw link of t link</para>
201    /// <para></para>
202    /// </returns>
203    [MethodImpl(MethodImplOptions.AggressiveInlining)]
204    protected virtual ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
205        ↳ AsRef<RawLink<TLinkAddress>>(Links + (RawLink<TLinkAddress>.SizeInBytes *
206        ↳ _addressToInt64Converter.Convert(link)));
207
208    /// <summary>
209    /// <para>
210    /// Gets the link values using the specified link index.
211    /// </para>
212    /// <para></para>
213    /// </summary>
214    /// <param name="linkIndex">
215    /// <para>The link index.</para>
216    /// <para></para>
217    /// </param>
218    /// <returns>
219    /// <para>A list of t link</para>
220    /// <para></para>
221    /// </returns>
222    [MethodImpl(MethodImplOptions.AggressiveInlining)]
223    protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
224    {
225        ref var link = ref GetLinkReference(linkIndex);
226        return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
227    }
228
229    /// <summary>
230    /// <para>
231    /// Determines whether this instance first is to the left of second.
232    /// </para>
233    /// <para></para>
234    /// </summary>
235    /// <param name="first">
236    /// <para>The first.</para>
237    /// <para></para>
238    /// </param>
239    /// <param name="second">
240    /// <para>The second.</para>
241    /// <para></para>
242    /// </param>
243    /// <returns>
244    /// <para>The bool</para>
245    /// <para></para>
246    /// </returns>
247    [MethodImpl(MethodImplOptions.AggressiveInlining)]
248    protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
249    {
250        ref var firstLink = ref GetLinkReference(first);
251        ref var secondLink = ref GetLinkReference(second);
252        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
253        ↳ secondLink.Source, secondLink.Target);
254    }
255
256    /// <summary>
257    /// <para>
258    /// Determines whether this instance first is to the right of second.

```

```

255     /// </para>
256     /// <para></para>
257     /// </summary>
258     /// <param name="first">
259     /// <para>The first.</para>
260     /// <para></para>
261     /// </param>
262     /// <param name="second">
263     /// <para>The second.</para>
264     /// <para></para>
265     /// </param>
266     /// <returns>
267     /// <para>The bool</para>
268     /// <para></para>
269     /// </returns>
270     [MethodImpl(MethodImplOptions.AggressiveInlining)]
271     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
        ↪ second)
272     {
273         ref var firstLink = ref GetLinkReference(first);
274         ref var secondLink = ref GetLinkReference(second);
275         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
276     }
277
278     /// <summary>
279     /// <para>
280     /// Gets the size value using the specified 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 link</para>
290     /// <para></para>
291     /// </returns>
292     [MethodImpl(MethodImplOptions.AggressiveInlining)]
293     protected virtual TLinkAddress GetSizeValue(TLinkAddress value) =>
        ↪ Bit<TLinkAddress>.PartialRead(value, 5, -5);
294
295     /// <summary>
296     /// <para>
297     /// Sets the size value using the specified stored value.
298     /// </para>
299     /// <para></para>
300     /// </summary>
301     /// <param name="storedValue">
302     /// <para>The stored value.</para>
303     /// <para></para>
304     /// </param>
305     /// <param name="size">
306     /// <para>The size.</para>
307     /// <para></para>
308     /// </param>
309     [MethodImpl(MethodImplOptions.AggressiveInlining)]
310     protected virtual void SetSizeValue(ref TLinkAddress storedValue, TLinkAddress size) =>
        ↪ storedValue = Bit<TLinkAddress>.PartialWrite(storedValue, size, 5, -5);
311
312     /// <summary>
313     /// <para>
314     /// Determines whether this instance get left is child value.
315     /// </para>
316     /// <para></para>
317     /// </summary>
318     /// <param name="value">
319     /// <para>The value.</para>
320     /// <para></para>
321     /// </param>
322     /// <returns>
323     /// <para>The bool</para>
324     /// <para></para>
325     /// </returns>
326     [MethodImpl(MethodImplOptions.AggressiveInlining)]
327     protected virtual bool GetLeftIsChildValue(TLinkAddress value)

```

```

328 {
329     unchecked
330     {
331         return _addressToBoolConverter.Convert(Bit<TLinkAddress>.PartialRead(value, 4,
332             ↪ 1));
333         //return !EqualityComparer.Equals(Bit<TLinkAddress>.PartialRead(value, 4, 1),
334             ↪ default);
335     }
336 }
337
338 /// <summary>
339 /// <para>
340 /// Sets the left is child value using the specified stored value.
341 /// </para>
342 /// <para></para>
343 /// </summary>
344 /// <param name="storedValue">
345 /// <para>The stored value.</para>
346 /// <para></para>
347 /// </param>
348 /// <param name="value">
349 /// <para>The value.</para>
350 /// <para></para>
351 /// </param>
352 [MethodImpl(MethodImplOptions.AggressiveInlining)]
353 protected virtual void SetLeftIsChildValue(ref TLinkAddress storedValue, bool value)
354 {
355     unchecked
356     {
357         var previousValue = storedValue;
358         var modified = Bit<TLinkAddress>.PartialWrite(previousValue,
359             ↪ _boolToAddressConverter.Convert(value), 4, 1);
360         storedValue = modified;
361     }
362 }
363
364 /// <summary>
365 /// <para>
366 /// Determines whether this instance get right is child value.
367 /// </para>
368 /// <para></para>
369 /// </summary>
370 /// <param name="value">
371 /// <para>The value.</para>
372 /// <para></para>
373 /// </param>
374 /// <returns>
375 /// <para>The bool</para>
376 /// <para></para>
377 /// </returns>
378 [MethodImpl(MethodImplOptions.AggressiveInlining)]
379 protected virtual bool GetRightIsChildValue(TLinkAddress value)
380 {
381     unchecked
382     {
383         return _addressToBoolConverter.Convert(Bit<TLinkAddress>.PartialRead(value, 3,
384             ↪ 1));
385         //return !EqualityComparer.Equals(Bit<TLinkAddress>.PartialRead(value, 3, 1),
386             ↪ default);
387     }
388 }
389
390 /// <summary>
391 /// <para>
392 /// Sets the right is child value using the specified stored value.
393 /// </para>
394 /// <para></para>
395 /// </summary>
396 /// <param name="storedValue">
397 /// <para>The stored value.</para>
398 /// <para></para>
399 /// </param>
400 /// <param name="value">
401 /// <para>The value.</para>
402 /// <para></para>
403 /// </param>
404 [MethodImpl(MethodImplOptions.AggressiveInlining)]
405 protected virtual void SetRightIsChildValue(ref TLinkAddress storedValue, bool value)

```

```

401 {
402     unchecked
403     {
404         var previousValue = storedValue;
405         var modified = Bit<TLinkAddress>.PartialWrite(previousValue,
406             ↪ _boolToAddressConverter.Convert(value), 3, 1);
407         storedValue = modified;
408     }
409 }
410
411 /// <summary>
412 /// <para>
413 /// Determines whether this instance is child.
414 /// </para>
415 /// <para></para>
416 /// </summary>
417 /// <param name="parent">
418 /// <para>The parent.</para>
419 /// <para></para>
420 /// </param>
421 /// <param name="possibleChild">
422 /// <para>The possible child.</para>
423 /// <para></para>
424 /// </param>
425 /// <returns>
426 /// <para>The bool</para>
427 /// <para></para>
428 /// </returns>
429 [MethodImpl(MethodImplOptions.AggressiveInlining)]
430 protected bool IsChild(TLinkAddress parent, TLinkAddress possibleChild)
431 {
432     var parentSize = GetSize(parent);
433     var childSize = GetSizeOrZero(possibleChild);
434     return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
435 }
436
437 /// <summary>
438 /// <para>
439 /// Gets the balance value using the specified stored value.
440 /// </para>
441 /// <para></para>
442 /// </summary>
443 /// <param name="storedValue">
444 /// <para>The stored value.</para>
445 /// <para></para>
446 /// </param>
447 /// <returns>
448 /// <para>The sbyte</para>
449 /// <para></para>
450 /// </returns>
451 [MethodImpl(MethodImplOptions.AggressiveInlining)]
452 protected virtual sbyte GetBalanceValue(TLinkAddress storedValue)
453 {
454     unchecked
455     {
456         var value =
457             ↪ _addressToInt32Converter.Convert(Bit<TLinkAddress>.PartialRead(storedValue,
458                 ↪ 0, 3));
459         value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
460             ↪ end of sbyte
461         return (sbyte)value;
462     }
463 }
464
465 /// <summary>
466 /// <para>
467 /// Sets the balance value using the specified stored value.
468 /// </para>
469 /// <para></para>
470 /// </summary>
471 /// <param name="storedValue">
472 /// <para>The stored value.</para>
473 /// <para></para>
474 /// </param>
475 /// <param name="value">
476 /// <para>The value.</para>
477 /// <para></para>
478 /// </param>

```

```

475 [MethodImpl(MethodImplOptions.AggressiveInlining)]
476 protected virtual void SetBalanceValue(ref TLinkAddress storedValue, sbyte value)
477 {
478     unchecked
479     {
480         var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
481             ↪ value & 3);
482         var modified = Bit<TLinkAddress>.PartialWrite(storedValue, packagedValue, 0, 3);
483         storedValue = modified;
484     }
485 }
486
487 /// <summary>
488 /// <para>
489 /// The zero.
490 /// </para>
491 /// <para></para>
492 /// </summary>
493 public TLinkAddress this[TLinkAddress index]
494 {
495     [MethodImpl(MethodImplOptions.AggressiveInlining)]
496     get
497     {
498         var root = GetTreeRoot();
499         if (GreaterOrEqualThan(index, GetSize(root)))
500         {
501             return Zero;
502         }
503         while (!EqualToZero(root))
504         {
505             var left = GetLeftOrDefault(root);
506             var leftSize = GetSizeOrZero(left);
507             if (LessThan(index, leftSize))
508             {
509                 root = left;
510                 continue;
511             }
512             if (AreEqual(index, leftSize))
513             {
514                 return root;
515             }
516             root = GetRightOrDefault(root);
517             index = Subtract(index, Increment(leftSize));
518         }
519         return Zero; // TODO: Impossible situation exception (only if tree structure
520             ↪ broken)
521     }
522 }
523
524 /// <summary>
525 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
526 ↪ (концом).
527 /// </summary>
528 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
529 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
530 /// <returns>Индекс искомой связи.</returns>
531 [MethodImpl(MethodImplOptions.AggressiveInlining)]
532 public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
533 {
534     var root = GetTreeRoot();
535     while (!EqualToZero(root))
536     {
537         ref var rootLink = ref GetLinkReference(root);
538         var rootSource = rootLink.Source;
539         var rootTarget = rootLink.Target;
540         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
541             ↪ node.Key < root.Key
542         {
543             root = GetLeftOrDefault(root);
544         }
545         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
546             ↪ node.Key > root.Key
547         {
548             root = GetRightOrDefault(root);
549         }
550         else // node.Key == root.Key
551         {
552             return root;
553         }
554     }
555 }

```

```

548     }
549 }
550 return Zero;
551 }
552
553 // TODO: Return indices range instead of references count
554 /// <summary>
555 /// <para>
556 /// Counts the usages using the specified link.
557 /// </para>
558 /// <para></para>
559 /// </summary>
560 /// <param name="link">
561 /// <para>The link.</para>
562 /// <para></para>
563 /// </param>
564 /// <returns>
565 /// <para>The link</para>
566 /// <para></para>
567 /// </returns>
568 [MethodImpl(MethodImplOptions.AggressiveInlining)]
569 public TLinkAddress CountUsages(TLinkAddress link)
570 {
571     var root = GetTreeRoot();
572     var total = GetSize(root);
573     var totalRightIgnore = Zero;
574     while (!EqualToZero(root))
575     {
576         var @base = GetBasePartValue(root);
577         if (LessOrEqualThan(@base, link))
578         {
579             root = GetRightOrDefault(root);
580         }
581         else
582         {
583             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
584             root = GetLeftOrDefault(root);
585         }
586     }
587     root = GetTreeRoot();
588     var totalLeftIgnore = Zero;
589     while (!EqualToZero(root))
590     {
591         var @base = GetBasePartValue(root);
592         if (GreaterOrEqualThan(@base, link))
593         {
594             root = GetLeftOrDefault(root);
595         }
596         else
597         {
598             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
599             root = GetRightOrDefault(root);
600         }
601     }
602     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
603 }
604
605 /// <summary>
606 /// <para>
607 /// Eaches the usage using the specified link.
608 /// </para>
609 /// <para></para>
610 /// </summary>
611 /// <param name="link">
612 /// <para>The link.</para>
613 /// <para></para>
614 /// </param>
615 /// <param name="handler">
616 /// <para>The handler.</para>
617 /// <para></para>
618 /// </param>
619 /// <returns>
620 /// <para>The continue.</para>
621 /// <para></para>
622 /// </returns>
623 [MethodImpl(MethodImplOptions.AggressiveInlining)]
624 public TLinkAddress EachUsage(TLinkAddress link, ReadHandler<TLinkAddress>? handler)
625

```

```

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 }

```

1.82 ./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         /// <summary>
61         /// <para>
62         /// Initializes a new <see cref="LinksRecursionlessSizeBalancedTreeMethodsBase"/>
63         ↪ instance.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         /// <param name="constants">
68         /// <para>A constants.</para>
69         /// <para></para>
70         /// </param>
71         /// <param name="links">
72         /// <para>A links.</para>
73         /// <para></para>
74         /// </param>
75         /// <param name="header">
76         /// <para>A header.</para>
77         /// <para></para>
78         /// </param>
79         [MethodImpl(MethodImplOptions.AggressiveInlining)]
80         protected LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
81         ↪ constants, byte* links, byte* header)
82         {
83             Links = links;
84             Header = header;
85             Break = constants.Break;
86             Continue = constants.Continue;
87         }
88
89         /// <summary>
90         /// <para>
91         /// 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 FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
241 {
242     ref var firstLink = ref GetLinkReference(first);
243     ref var secondLink = ref GetLinkReference(second);
244     return FirstIsToTheLeftOfSecond(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
311    ↪ (концом).
312    /// </summary>
313    /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
314    /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
315    /// <returns>Индекс искомой связи.</returns>
316    [MethodImpl(MethodImplOptions.AggressiveInlining)]
317    public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
318    {
319        var root = GetTreeRoot();
320        while (!EqualToZero(root))
321        {
322            ref var rootLink = ref GetLinkReference(root);
323            var rootSource = rootLink.Source;
324            var rootTarget = rootLink.Target;
325            if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
326                ↪ node.Key < root.Key
327            {
328                root = GetLeftOrDefault(root);
329            }
330            else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
331                ↪ node.Key > root.Key
332            {
333                root = GetRightOrDefault(root);
334            }
335            else // node.Key == root.Key
336            {
337                return root;
338            }
339        }
340        return Zero;
341    }
342
343    /// TODO: Return indices range instead of references count
344    /// <summary>
345    /// <para>
346    /// Counts the usages using the specified link.
347    /// </para>
348    /// <para></para>
349    /// </summary>
350    /// <param name="link">
351    /// <para>The link.</para>
352    /// <para></para>
353    /// </param>
354    /// <returns>
355    /// <para>The link</para>
356    /// <para></para>
357    /// </returns>
358    [MethodImpl(MethodImplOptions.AggressiveInlining)]
359    public TLinkAddress CountUsages(TLinkAddress link)
360    {
361        var root = GetTreeRoot();
362        var total = GetSize(root);
363        var totalRightIgnore = Zero;
364        while (!EqualToZero(root))
365        {
366            var @base = GetBasePartValue(root);
367            if (LessOrEqualThan(@base, link))
368            {
369                root = GetRightOrDefault(root);
370            }
371            else
372            {
373                totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
374                root = GetLeftOrDefault(root);
375            }
376        }
377        root = GetTreeRoot();
378        var totalLeftIgnore = Zero;
379        while (!EqualToZero(root))
380        {
381            var @base = GetBasePartValue(root);
382            if (GreaterOrEqualThan(@base, link))
383            {
384                root = GetLeftOrDefault(root);
385            }
386            else
387            {
388                totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
389                root = GetRightOrDefault(root);
390            }
391        }
392        return Add(total, totalLeftIgnore, totalRightIgnore);
393    }

```

```

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/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>
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 }

```

```

54     /// <summary>
55     /// <para>
56     /// Initializes a new <see cref="LinksSizeBalancedTreeMethodsBase"/> instance.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     /// <param name="constants">
61     /// <para>A constants.</para>
62     /// <para></para>
63     /// </param>
64     /// <param name="links">
65     /// <para>A links.</para>
66     /// <para></para>
67     /// </param>
68     /// <param name="header">
69     /// <para>A header.</para>
70     /// <para></para>
71     /// </param>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected LinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, byte*
74     ↪ 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 FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
241 {
242     ref var firstLink = ref GetLinkReference(first);
243     ref var secondLink = ref GetLinkReference(second);
244     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
245         ↪ secondLink.Source, secondLink.Target);
246 }
247
248 /// <summary>
249 /// <para>
250 /// Determines whether this instance first is to the right of second.
251 /// </para>
252 /// <para></para>
253 /// </summary>
254 /// <param name="first">
255 /// <para>The first.</para>
256 /// <para></para>
257 /// </param>
258 /// <param name="second">
259 /// <para>The second.</para>
260 /// <para></para>
261 /// </param>
262 /// <returns>
263 /// <para>The bool</para>
264 /// <para></para>
265 /// </returns>
266 [MethodImpl(MethodImplOptions.AggressiveInlining)]
267 protected override bool FirstIsToTheRightOfSecond(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 }

```

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

```

```

170 [MethodImpl(MethodImplOptions.AggressiveInlining)]
171 protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
    ↳ SetSizeValue(ref GetLinkReference(node).SizeAsSource, size);

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

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

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

223
224 /// <summary>
225 /// <para>
226 /// Sets the right is child using the specified node.
227 /// </para>
228 /// <para></para>
229 /// </summary>
230 /// <param name="node">
231 /// <para>The node.</para>
232 /// <para></para>
233 /// </param>
234 /// <param name="value">
235 /// <para>The value.</para>
236 /// <para></para>
237 /// </param>
238 [MethodImpl(MethodImplOptions.AggressiveInlining)]
239 protected override void SetRightIsChild(TLinkAddress node, bool value) =>
    ↳ SetRightIsChildValue(ref GetLinkReference(node).SizeAsSource, value);

240
241 /// <summary>

```



```

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

```

```

317     /// <para></para>
318     /// </param>
319     /// <param name="secondSource">
320     /// <para>The second source.</para>
321     /// <para></para>
322     /// </param>
323     /// <param name="secondTarget">
324     /// <para>The second target.</para>
325     /// <para></para>
326     /// </param>
327     /// <returns>
328     /// <para>The bool</para>
329     /// <para></para>
330     /// </returns>
331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
332     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
        ↪ LessThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
        ↪ LessThan(firstTarget, secondTarget));
333
334     /// <summary>
335     /// <para>
336     /// Determines whether this instance first is to the right of second.
337     /// </para>
338     /// <para></para>
339     /// </summary>
340     /// <param name="firstSource">
341     /// <para>The first source.</para>
342     /// <para></para>
343     /// </param>
344     /// <param name="firstTarget">
345     /// <para>The first target.</para>
346     /// <para></para>
347     /// </param>
348     /// <param name="secondSource">
349     /// <para>The second source.</para>
350     /// <para></para>
351     /// </param>
352     /// <param name="secondTarget">
353     /// <para>The second target.</para>
354     /// <para></para>
355     /// </param>
356     /// <returns>
357     /// <para>The bool</para>
358     /// <para></para>
359     /// </returns>
360     [MethodImpl(MethodImplOptions.AggressiveInlining)]
361     protected override bool FirstIsToTheRightOfSecond(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 }

```

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

1 using System.Runtime.CompilerServices;

2

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

4

```

5 namespace Platform.Data.Doublets.Memory.United.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
14    public unsafe class LinksSourcesRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
15        ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="LinksSourcesRecursionlessSizeBalancedTreeMethods"/>
20        ↳ instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// </param>
30        /// <param name="header">
31        /// <para>A header.</para>
32        /// </param>
33        /// </param>
34        [MethodImpl(MethodImplOptions.AggressiveInlining)]
35        public LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
36            ↳ constants, byte* links, byte* header) : base(constants, links, header) { }
37
38        /// <summary>
39        /// <para>
40        /// Gets the left reference using the specified node.
41        /// </para>
42        /// <para></para>
43        /// </summary>
44        /// <param name="node">
45        /// <para>The node.</para>
46        /// </param>
47        /// <returns>
48        /// <para>The ref link</para>
49        /// </returns>
50        [MethodImpl(MethodImplOptions.AggressiveInlining)]
51        protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
52            ↳ 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        /// </param>
63        /// <returns>
64        /// <para>The ref link</para>
65        /// </returns>
66        [MethodImpl(MethodImplOptions.AggressiveInlining)]
67        protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
68            ↳ GetLinkReference(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">

```

```

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).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 link</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected override TLinkAddress GetRight(TLinkAddress node) =>
105        ↪ GetLinkReference(node).RightAsSource;
106
107    /// <summary>
108    /// <para>
109    /// Sets the left using the specified node.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="node">
114    /// <para>The node.</para>
115    /// <para></para>
116    /// </param>
117    /// <param name="left">
118    /// <para>The left.</para>
119    /// <para></para>
120    /// </param>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
123        ↪ GetLinkReference(node).LeftAsSource = 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).RightAsSource = right;
142
143    /// <summary>
144    /// <para>
145    /// Gets the size using the specified node.
146    /// </para>
147    /// <para></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>

```

```

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     /// </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         ↪ GetLinkReference(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     /// </param>
191     /// <returns>
192     /// <para>The link</para>
193     /// </returns>
194     [MethodImpl(MethodImplOptions.AggressiveInlining)]
195     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
196         ↪ GetLinkReference(link).Source;
197
198     /// <summary>
199     /// <para>
200     /// Determines whether this instance first is to the left of second.
201     /// </para>
202     /// </summary>
203     /// <param name="firstSource">
204     /// <para>The first source.</para>
205     /// </param>
206     /// <param name="firstTarget">
207     /// <para>The first target.</para>
208     /// </param>
209     /// <param name="secondSource">
210     /// <para>The second source.</para>
211     /// </param>
212     /// <param name="secondTarget">
213     /// <para>The second target.</para>
214     /// </param>
215     /// <returns>
216     /// <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/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> :
        ↪ LinksSizeBalancedTreeMethodsBase<TLinkAddress>
15    {

```

```

16    /// <summary>
17    /// <para>
18    /// Initializes a new <see cref="LinksSourcesSizeBalancedTreeMethods"/> instance.
19    /// </para>
20    /// <para></para>
21    /// </summary>
22    /// <param name="constants">
23    /// <para>A constants.</para>
24    /// <para></para>
25    /// </param>
26    /// <param name="links">
27    /// <para>A links.</para>
28    /// <para></para>
29    /// </param>
30    /// <param name="header">
31    /// <para>A header.</para>
32    /// <para></para>
33    /// </param>
34    [MethodImpl(MethodImplOptions.AggressiveInlining)]
35    public LinksSourcesSizeBalancedTreeMethods(LinksConstants<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) =>
104        ↪ GetLinkReference(node).RightAsSource;
105
106    /// <summary>
107    /// <para>
108    /// Sets the left using the specified node.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <param name="node">
113    /// <para>The node.</para>
114    /// <para></para>
115    /// </param>
116    /// <param name="left">
117    /// <para>The left.</para>
118    /// <para></para>
119    /// </param>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
122        ↪ GetLinkReference(node).LeftAsSource = left;
123
124    /// <summary>
125    /// <para>
126    /// Sets the right using the specified node.
127    /// </para>
128    /// <para></para>
129    /// </summary>
130    /// <param name="node">
131    /// <para>The node.</para>
132    /// <para></para>
133    /// </param>
134    /// <param name="right">
135    /// <para>The right.</para>
136    /// <para></para>
137    /// </param>
138    [MethodImpl(MethodImplOptions.AggressiveInlining)]
139    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
140        ↪ GetLinkReference(node).RightAsSource = right;
141
142    /// <summary>
143    /// <para>
144    /// Gets the size using the specified node.
145    /// </para>
146    /// <para></para>
147    /// </summary>
148    /// <param name="node">
149    /// <para>The node.</para>
150    /// <para></para>
151    /// </param>
152    /// <returns>
153    /// <para>The link</para>
154    /// <para></para>
155    /// </returns>
156    [MethodImpl(MethodImplOptions.AggressiveInlining)]
157    protected override TLinkAddress GetSize(TLinkAddress node) =>
158        ↪ GetLinkReference(node).SizeAsSource;
159
160    /// <summary>
161    /// <para>
162    /// Sets the size using the specified node.
163    /// </para>
164    /// <para></para>
165    /// </summary>
166    /// <param name="node">
167    /// <para>The node.</para>

```



```

164     /// <para></para>
165     /// </param>
166     /// <param name="size">
167     /// <para>The size.</para>
168     /// <para></para>
169     /// </param>
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
172         ↪ GetLinkReference(node).SizeAsSource = size;
173
174     /// <summary>
175     /// <para>
176     /// Gets the tree root.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <returns>
181     /// <para>The link</para>
182     /// <para></para>
183     /// </returns>
184     [MethodImpl(MethodImplOptions.AggressiveInlining)]
185     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsSource;
186
187     /// <summary>
188     /// <para>
189     /// Gets the base part value using the specified link.
190     /// </para>
191     /// <para></para>
192     /// </summary>
193     /// <param name="link">
194     /// <para>The link.</para>
195     /// <para></para>
196     /// </param>
197     /// <returns>
198     /// <para>The link</para>
199     /// <para></para>
200     /// </returns>
201     [MethodImpl(MethodImplOptions.AggressiveInlining)]
202     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
203         ↪ GetLinkReference(link).Source;
204
205     /// <summary>
206     /// <para>
207     /// Determines whether this instance first is to the left of second.
208     /// </para>
209     /// <para></para>
210     /// </summary>
211     /// <param name="firstSource">
212     /// <para>The first source.</para>
213     /// <para></para>
214     /// </param>
215     /// <param name="firstTarget">
216     /// <para>The first target.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="secondSource">
220     /// <para>The second source.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="secondTarget">
224     /// <para>The second target.</para>
225     /// <para></para>
226     /// </param>
227     /// <returns>
228     /// <para>The bool</para>
229     /// <para></para>
230     /// </returns>
231     [MethodImpl(MethodImplOptions.AggressiveInlining)]
232     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
233         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
234         ↪ LessThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
235         ↪ LessThan(firstTarget, secondTarget));
236
237     /// <summary>
238     /// <para>
239     /// Determines whether this instance first is to the right of second.
240     /// </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     /// <summary>
261     /// <para>
262     /// <para>Clears the node using the specified node.
263     /// </para>
264     /// <para></para>
265     /// </summary>
266     /// <param name="node">
267     /// <para>The node.</para>
268     /// <para></para>
269     /// </param>
270     [MethodImpl(MethodImplOptions.AggressiveInlining)]
271     protected override void ClearNode(TLinkAddress node)
272     {
273         ref var link = ref GetLinkReference(node);
274         link.LeftAsSource = Zero;
275         link.RightAsSource = Zero;
276         link.SizeAsSource = Zero;
277     }
278 }
279 }
280 }

```

1.87 ./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     /// <para>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> :
        ↪ LinksAvlBalancedTreeMethodsBase<TLinkAddress>
15    {
16        /// <summary>
17        /// <para>
18        /// <para>Initializes a new <see cref="LinksTargetsAvlBalancedTreeMethods"/> instance.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <param name="constants">
23        /// <para>A constants.</para>
24        /// <para></para>
25        /// </param>
26        /// <param name="links">
27        /// <para>A links.</para>

```

```

28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public LinkTargetsAvlBalancedTreeMethods(LinkConstants<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).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) =>
    ↳ GetLinkReference(node).RightAsTarget = right;
138
139 /// <summary>
140 /// <para>
141 /// Gets the size using the specified node.
142 /// </para>
143 /// <para></para>
144 /// </summary>
145 /// <param name="node">
146 /// <para>The node.</para>
147 /// <para></para>
148 /// </param>
149 /// <returns>
150 /// <para>The link</para>
151 /// <para></para>
152 /// </returns>
153 [MethodImpl(MethodImplOptions.AggressiveInlining)]
154 protected override TLinkAddress GetSize(TLinkAddress node) =>
    ↳ GetSizeValue(GetLinkReference(node).SizeAsTarget);
155
156 /// <summary>
157 /// <para>
158 /// Sets the size using the specified node.
159 /// </para>
160 /// <para></para>
161 /// </summary>
162 /// <param name="node">
163 /// <para>The node.</para>
164 /// <para></para>
165 /// </param>
166 /// <param name="size">
167 /// <para>The size.</para>
168 /// <para></para>
169 /// </param>
170 [MethodImpl(MethodImplOptions.AggressiveInlining)]
171 protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
    ↳ SetSizeValue(ref GetLinkReference(node).SizeAsTarget, size);
172
173 /// <summary>

```

```

174    /// <para>
175    /// Determines whether this instance get left is child.
176    /// </para>
177    /// <para></para>
178    /// </summary>
179    /// <param name="node">
180    /// <para>The node.</para>
181    /// <para></para>
182    /// </param>
183    /// <returns>
184    /// <para>The bool</para>
185    /// <para></para>
186    /// </returns>
187    [MethodImpl(MethodImplOptions.AggressiveInlining)]
188    protected override bool GetLeftIsChild(TLinkAddress node) =>
189        ↪ GetLeftIsChildValue(GetLinkReference(node).SizeAsTarget);
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(TLinkAddress node, bool value) =>
207        ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsTarget, 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(TLinkAddress node) =>
225        ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsTarget);
226
227    /// <summary>
228    /// <para>
229    /// Sets the right is child using the specified node.
230    /// </para>
231    /// <para></para>
232    /// </summary>
233    /// <param name="node">
234    /// <para>The node.</para>
235    /// <para></para>
236    /// </param>
237    /// <param name="value">
238    /// <para>The value.</para>
239    /// <para></para>
240    /// </param>
241    [MethodImpl(MethodImplOptions.AggressiveInlining)]
242    protected override void SetRightIsChild(TLinkAddress node, bool value) =>
243        ↪ SetRightIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
244
245    /// <summary>
246    /// <para>
247    /// Gets the balance using the specified node.
248    /// </para>
249    /// <para></para>
250    /// </summary>
251    /// <param name="node">

```

```

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

```

```

323     /// <param name="secondTarget">
324     /// <para>The second target.</para>
325     /// </para>
326     /// </param>
327     /// <returns>
328     /// <para>The bool</para>
329     /// <para></para>
330     /// </returns>
331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
332     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
        ↪ LessThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
        ↪ LessThan(firstSource, secondSource));

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

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

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links targets recursionless size balanced tree methods.
10    /// </para>

```

```

11  /// <para></para>
12  /// </summary>
13  /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
14  public unsafe class LinksTargetsRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
    ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
15  {
16      /// <summary>
17      /// <para>
18      /// Initializes a new <see cref="LinksTargetsRecursionlessSizeBalancedTreeMethods"/>
19      ↳ instance.
20      /// </para>
21      /// <para></para>
22      /// </summary>
23      /// <param name="constants">
24      /// <para>A constants.</para>
25      /// </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 LinksTargetsRecursionlessSizeBalancedTreeMethods(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).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) =>
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>
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        ↪ GetLinkReference(node).SizeAsTarget;

```

```

156    /// <summary>
157    /// <para>
158    /// Sets the size using the specified node.
159    /// </para>
160    /// <para></para>
161    /// </summary>
162    /// <param name="node">
163    /// <para>The node.</para>
164    /// <para></para>
165    /// </param>
166    /// <param name="size">
167    /// <para>The size.</para>
168    /// <para></para>
169    /// </param>
170    [MethodImpl(MethodImplOptions.AggressiveInlining)]
171    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
172        ↪ GetLinkReference(node).SizeAsTarget = size;
173
174    /// <summary>
175    /// <para>
176    /// Gets the tree root.
177    /// </para>
178    /// <para></para>
179    /// </summary>
180    /// <returns>
181    /// <para>The link</para>
182    /// <para></para>
183    /// </returns>
184    [MethodImpl(MethodImplOptions.AggressiveInlining)]
185    protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsTarget;
186
187    /// <summary>
188    /// <para>
189    /// Gets the base part value using the specified link.
190    /// </para>
191    /// <para></para>
192    /// </summary>
193    /// <param name="link">
194    /// <para>The link.</para>
195    /// <para></para>
196    /// </param>
197    /// <returns>
198    /// <para>The link</para>
199    /// <para></para>
200    /// </returns>
201    [MethodImpl(MethodImplOptions.AggressiveInlining)]
202    protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
203        ↪ GetLinkReference(link).Target;
204
205    /// <summary>
206    /// <para>
207    /// Determines whether this instance first is to the left of second.
208    /// </para>
209    /// <para></para>
210    /// </summary>
211    /// <param name="firstSource">
212    /// <para>The first source.</para>
213    /// <para></para>
214    /// </param>
215    /// <param name="firstTarget">
216    /// <para>The first target.</para>
217    /// <para></para>
218    /// </param>
219    /// <param name="secondSource">
220    /// <para>The second source.</para>
221    /// <para></para>
222    /// </param>
223    /// <param name="secondTarget">
224    /// <para>The second target.</para>
225    /// <para></para>
226    /// </param>
227    /// <returns>
228    /// <para>The bool</para>
229    /// <para></para>
230    /// </returns>
231    [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/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> :
        ↪ LinksSizeBalancedTreeMethodsBase<TLinkAddress>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="LinksTargetsSizeBalancedTreeMethods"/> instance.

```

```

19    /// </para>
20    /// <para></para>
21    /// </summary>
22    /// <param name="constants">
23    /// <para>A constants.</para>
24    /// <para></para>
25    /// </param>
26    /// <param name="links">
27    /// <para>A links.</para>
28    /// <para></para>
29    /// </param>
30    /// <param name="header">
31    /// <para>A header.</para>
32    /// <para></para>
33    /// </param>
34    [MethodImpl(MethodImplOptions.AggressiveInlining)]
35    public LinksTargetsSizeBalancedTreeMethods(LinksConstants<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).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) =>
104        ↪ GetLinkReference(node).RightAsTarget;
105
106    /// <summary>
107    /// <para>
108    /// Sets the left using the specified node.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <param name="node">
113    /// <para>The node.</para>
114    /// <para></para>
115    /// </param>
116    /// <param name="left">
117    /// <para>The left.</para>
118    /// <para></para>
119    /// </param>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
122        ↪ GetLinkReference(node).LeftAsTarget = left;
123
124    /// <summary>
125    /// <para>
126    /// Sets the right using the specified node.
127    /// </para>
128    /// <para></para>
129    /// </summary>
130    /// <param name="node">
131    /// <para>The node.</para>
132    /// <para></para>
133    /// </param>
134    /// <param name="right">
135    /// <para>The right.</para>
136    /// <para></para>
137    /// </param>
138    [MethodImpl(MethodImplOptions.AggressiveInlining)]
139    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
140        ↪ GetLinkReference(node).RightAsTarget = right;
141
142    /// <summary>
143    /// <para>
144    /// Gets the size using the specified node.
145    /// </para>
146    /// <para></para>
147    /// </summary>
148    /// <param name="node">
149    /// <para>The node.</para>
150    /// <para></para>
151    /// </param>
152    /// <returns>
153    /// <para>The link</para>
154    /// <para></para>
155    /// </returns>
156    [MethodImpl(MethodImplOptions.AggressiveInlining)]
157    protected override TLinkAddress GetSize(TLinkAddress node) =>
158        ↪ GetLinkReference(node).SizeAsTarget;
159
160    /// <summary>
161    /// <para>
162    /// Sets the size using the specified node.
163    /// </para>
164    /// <para></para>
165    /// </summary>
166    /// <param name="node">
167    /// <para>The node.</para>
168    /// <para></para>
169    /// </param>
170    /// <param name="size">

```

```

167     /// <para>The size.</para>
168     /// <para></para>
169     /// </param>
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
172         ↪ GetLinkReference(node).SizeAsTarget = size;
173
174     /// <summary>
175     /// <para>
176     /// Gets the tree root.
177     /// </para>
178     /// </summary>
179     /// <returns>
180     /// <para>The link</para>
181     /// <para></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     /// <para></para>
191     /// </summary>
192     /// <param name="link">
193     /// <para>The link.</para>
194     /// <para></para>
195     /// </param>
196     /// <returns>
197     /// <para>The link</para>
198     /// <para></para>
199     /// </returns>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     protected override 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 FirstIsToTheLeftOfSecond(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">

```

```

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 }

```

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

```



```

101     if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
102     {
103         _createSourceTreeMethods = () => new
104             ↳ LinksSourcesAvlBalancedTreeMethods<TLinkAddress>(Constants, _links, _header);
105         _createTargetTreeMethods = () => new
106             ↳ LinksTargetsAvlBalancedTreeMethods<TLinkAddress>(Constants, _links, _header);
107     }
108     else
109     {
110         _createSourceTreeMethods = () => new
111             ↳ LinksSourcesSizeBalancedTreeMethods<TLinkAddress>(Constants, _links,
112             ↳ _header);
113         _createTargetTreeMethods = () => new
114             ↳ LinksTargetsSizeBalancedTreeMethods<TLinkAddress>(Constants, _links,
115             ↳ _header);
116     }
117     Init(memory, memoryReservationStep);
118 }
119
120 /// <summary>
121 /// <para>
122 /// Sets the pointers using the specified memory.
123 /// </para>
124 /// <para></para>
125 /// </summary>
126 /// <param name="memory">
127 /// <para>The memory.</para>
128 /// <para></para>
129 /// </param>
130 [MethodImpl(MethodImplOptions.AggressiveInlining)]
131 protected override void SetPointers(IResizableDirectMemory memory)
132 {
133     _links = (byte*)memory.Pointer;
134     _header = _links;
135     SourcesTreeMethods = _createSourceTreeMethods();
136     TargetsTreeMethods = _createTargetTreeMethods();
137     UnusedLinksListMethods = new UnusedLinksListMethods<TLinkAddress>(_links, _header);
138 }
139
140 /// <summary>
141 /// <para>
142 /// Resets the pointers.
143 /// </para>
144 /// <para></para>
145 /// </summary>
146 [MethodImpl(MethodImplOptions.AggressiveInlining)]
147 protected override void ResetPointers()
148 {
149     base.ResetPointers();
150     _links = null;
151     _header = null;
152 }
153
154 /// <summary>
155 /// <para>
156 /// Gets the header reference.
157 /// </para>
158 /// <para></para>
159 /// </summary>
160 /// <returns>
161 /// <para>A ref links header of t link</para>
162 /// <para></para>
163 /// </returns>
164 [MethodImpl(MethodImplOptions.AggressiveInlining)]
165 protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
166     ↳ AsRef<LinksHeader<TLinkAddress>>(_header);
167
168 /// <summary>
169 /// <para>
170 /// Gets the link reference using the specified link index.
171 /// </para>
172 /// <para></para>
173 /// </summary>
174 /// <param name="linkIndex">
175 /// <para>The link index.</para>
176 /// <para></para>
177 /// </param>
178 /// <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 }

```

1.91 ./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>
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 =
        ↪ LinksHeader<TLinkAddress>.SizeInBytes;
47
48         /// <summary>
49         /// <para>
50         /// The link size in bytes.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
55
56         /// <summary>
57         /// <para>
58         /// The memory.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         protected readonly IResizableDirectMemory _memory;
63         /// <summary>

```

```

64     /// <para>
65     /// The memory reservation step.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     protected readonly long _memoryReservationStep;
70
71     /// <summary>
72     /// <para>
73     /// The targets tree methods.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     protected ILinksTreeMethods<TLinkAddress> TargetsTreeMethods;
78     /// <summary>
79     /// <para>
80     /// The sources tree methods.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     protected ILinksTreeMethods<TLinkAddress> SourcesTreeMethods;
85     // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
86     // → нужно использовать не список а дерево, так как так можно быстрее проверить на
87     // → наличие связи внутри
88     /// <summary>
89     /// <para>
90     /// The unused links list methods.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     protected ILinksListMethods<TLinkAddress> UnusedLinksListMethods;
95
96     /// <summary>
97     /// Возвращает общее число связей находящихся в хранилище.
98     /// </summary>
99     protected virtual TLinkAddress Total
100     {
101         [MethodImpl(MethodImplOptions.AggressiveInlining)]
102         get
103         {
104             ref var header = ref GetHeaderReference();
105             return Subtract(header.AllocatedLinks, header.FreeLinks);
106         }
107     }
108
109     /// <summary>
110     /// <para>
111     /// Gets the constants value.
112     /// </para>
113     /// <para></para>
114     /// </summary>
115     public virtual LinksConstants<TLinkAddress> Constants
116     {
117         [MethodImpl(MethodImplOptions.AggressiveInlining)]
118         get;
119     }
120
121     /// <summary>
122     /// <para>
123     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
124     /// </para>
125     /// <para></para>
126     /// </summary>
127     /// <param name="memory">
128     /// <para>A memory.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="memoryReservationStep">
132     /// <para>A memory reservation step.</para>
133     /// <para></para>
134     /// </param>
135     /// <param name="constants">
136     /// <para>A constants.</para>
137     /// <para></para>
138     /// </param>
139     [MethodImpl(MethodImplOptions.AggressiveInlining)]
140     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
141     // → memoryReservationStep, LinksConstants<TLinkAddress> constants)
142     {

```

```

140     _memory = memory;
141     _memoryReservationStep = memoryReservationStep;
142     Constants = constants;
143 }
144
145 /// <summary>
146 /// <para>
147 /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <param name="memory">
152 /// <para>A memory.</para>
153 /// <para></para>
154 /// </param>
155 /// <param name="memoryReservationStep">
156 /// <para>A memory reservation step.</para>
157 /// <para></para>
158 /// </param>
159 [MethodImpl(MethodImplOptions.AggressiveInlining)]
160 protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
    ↪ memoryReservationStep) : this(memory, memoryReservationStep,
    ↪ Default<LinksConstants<TLinkAddress>>.Instance) { }
161
162 /// <summary>
163 /// <para>
164 /// Inits the memory.
165 /// </para>
166 /// <para></para>
167 /// </summary>
168 /// <param name="memory">
169 /// <para>The memory.</para>
170 /// <para></para>
171 /// </param>
172 /// <param name="memoryReservationStep">
173 /// <para>The memory reservation step.</para>
174 /// <para></para>
175 /// </param>
176 [MethodImpl(MethodImplOptions.AggressiveInlining)]
177 protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
178 {
179     if (memory.ReservedCapacity < memoryReservationStep)
180     {
181         memory.ReservedCapacity = memoryReservationStep;
182     }
183     SetPointers(memory);
184     ref var header = ref GetHeaderReference();
185     // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
186     memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
    ↪ LinkHeaderSizeInBytes;
187     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
188     header.ReservedLinks = ConvertToAddress((memory.ReservedCapacity -
    ↪ LinkHeaderSizeInBytes) / LinkSizeInBytes);
189 }
190
191 /// <summary>
192 /// <para>
193 /// Counts the substitution.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <param name="restriction">
198 /// <para>The substitution.</para>
199 /// <para></para>
200 /// </param>
201 /// <exception cref="NotSupportedException">
202 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
203 /// <para></para>
204 /// </exception>
205 /// <returns>
206 /// <para>The link</para>
207 /// <para></para>
208 /// </returns>
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 public virtual TLinkAddress Count(IList<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 = this.GetIndex(restriction);
220 if (restriction.Count == 1)
221 {
222     if (AreEqual(index, any))
223     {
224         return Total;
225     }
226     return Exists(index) ? GetOne() : GetZero();
227 }
228 if (restriction.Count == 2)
229 {
230     var value = restriction[1];
231     if (AreEqual(index, any))
232     {
233         if (AreEqual(value, any))
234         {
235             return Total; // Any - как отсутствие ограничения
236         }
237         return Add(SourcesTreeMethods.CountUsages(value),
238             ↪ TargetsTreeMethods.CountUsages(value));
239     }
240     else
241     {
242         if (!Exists(index))
243         {
244             return GetZero();
245         }
246         if (AreEqual(value, any))
247         {
248             return GetOne();
249         }
250         ref var storedLinkValue = ref GetLinkReference(index);
251         if (AreEqual(storedLinkValue.Source, value) ||
252             ↪ AreEqual(storedLinkValue.Target, value))
253         {
254             return GetOne();
255         }
256         return GetZero();
257     }
258 }
259 if (restriction.Count == 3)
260 {
261     var source = this.GetSource(restriction);
262     var target = this.GetTarget(restriction);
263     if (AreEqual(index, any))
264     {
265         if (AreEqual(source, any) && AreEqual(target, any))
266         {
267             return Total;
268         }
269         else if (AreEqual(source, any))
270         {
271             return TargetsTreeMethods.CountUsages(target);
272         }
273         else if (AreEqual(target, any))
274         {
275             return SourcesTreeMethods.CountUsages(source);
276         }
277         else //if(source != Any && target != Any)
278         {
279             // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
280             var link = SourcesTreeMethods.Search(source, target);
281             return AreEqual(link, constants.Null) ? GetZero() : GetOne();
282         }
283     }
284     else
285     {
286         if (!Exists(index))
287         {
288             return GetZero();
289         }
290         if (AreEqual(source, any) && AreEqual(target, any))
291         {

```

```

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
320 /// <summary>
321 /// <para>
322 /// Eaches the handler.
323 /// </para>
324 /// <para></para>
325 /// </summary>
326 /// <param name="handler">
327 /// <para>The handler.</para>
328 /// <para></para>
329 /// </param>
330 /// <param name="restriction">
331 /// <para>The substitution.</para>
332 /// <para></para>
333 /// </param>
334 /// <exception cref="NotSupportedException">
335 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
336 /// <para></para>
337 /// </exception>
338 /// <returns>
339 /// <para>The link</para>
340 /// <para></para>
341 /// </returns>
342 [MethodImpl(MethodImplOptions.AggressiveInlining)]
343 public virtual TLinkAddress Each(IList<TLinkAddress>? restriction,
344     ↪ ReadHandler<TLinkAddress>? handler)
345 {
346     var constants = Constants;
347     var @break = constants.Break;
348     if (restriction.Count == 0)
349     {
350         for (var link = GetOne(); LessOrEqualThan(link,
351             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
352         {
353             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
354             {
355                 return @break;
356             }
357         }
358         return @break;
359     }
360     var @continue = constants.Continue;
361     var any = constants.Any;
362     var index = this.GetIndex(restriction);
363     if (restriction.Count == 1)
364     {

```

```

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),
383             ↪ @break))
384         {
385             return @break;
386         }
387         return Each(new Link<TLinkAddress>(index, any, value), handler);
388     }
389     else
390     {
391         if (!Exists(index))
392         {
393             return @continue;
394         }
395         if (AreEqual(value, any))
396         {
397             return handler(GetLinkStruct(index));
398         }
399         ref var storedLinkValue = ref GetLinkReference(index);
400         if (AreEqual(storedLinkValue.Source, value) ||
401             AreEqual(storedLinkValue.Target, value))
402         {
403             return handler(GetLinkStruct(index));
404         }
405         return @continue;
406     }
407 }
408 if (restriction.Count == 3)
409 {
410     var source = this.GetSource(restriction);
411     var target = this.GetTarget(restriction);
412     if (AreEqual(index, any))
413     {
414         if (AreEqual(source, any) && AreEqual(target, any))
415         {
416             return Each(Array.Empty<TLinkAddress>(), handler);
417         }
418         else if (AreEqual(source, any))
419         {
420             return TargetsTreeMethods.EachUsage(target, handler);
421         }
422         else if (AreEqual(target, any))
423         {
424             return SourcesTreeMethods.EachUsage(source, handler);
425         }
426         else //if(source != Any && target != Any)
427         {
428             var link = SourcesTreeMethods.Search(source, target);
429             return AreEqual(link, constants.Null) ? @continue :
430                 ↪ handler(GetLinkStruct(link));
431         }
432     }
433     else
434     {
435         if (!Exists(index))
436         {
437             return @continue;
438         }
439         if (AreEqual(source, any) && AreEqual(target, any))
440         {

```

```

439         return handler(GetLinkStruct(index));
440     }
441     ref var storedLinkValue = ref GetLinkReference(index);
442     if (!AreEqual(source, any) && !AreEqual(target, any))
443     {
444         if (AreEqual(storedLinkValue.Source, source) &&
445             AreEqual(storedLinkValue.Target, target))
446         {
447             return handler(GetLinkStruct(index));
448         }
449         return @continue;
450     }
451     var value = default(TLinkAddress);
452     if (AreEqual(source, any))
453     {
454         value = target;
455     }
456     if (AreEqual(target, any))
457     {
458         value = source;
459     }
460     if (AreEqual(storedLinkValue.Source, value) ||
461         AreEqual(storedLinkValue.Target, value))
462     {
463         return handler(GetLinkStruct(index));
464     }
465     return @continue;
466 }
467 throw new NotSupportedException("Другие размеры и способы ограничений не
468     ↳ поддерживаются.");
469 }
470
471 /// <remarks>
472 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
473     ↳ в другом месте (но не в менеджере памяти, а в логике Links)
474 /// </remarks>
475 [MethodImpl(MethodImplOptions.AggressiveInlining)]
476 public virtual TLinkAddress Update(IList<TLinkAddress>? restriction,
477     ↳ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
478 {
479     var constants = Constants;
480     var @null = constants.Null;
481     var linkIndex = this.GetIndex(restriction);
482     var before = GetLinkStruct(linkIndex);
483     ref var link = ref GetLinkReference(linkIndex);
484     ref var header = ref GetHeaderReference();
485     ref var firstAsSource = ref header.RootAsSource;
486     ref var firstAsTarget = ref header.RootAsTarget;
487     // Будет корректно работать только в том случае, если пространство выделенной связи
488     ↳ предварительно заполнено нулями
489     if (!AreEqual(link.Source, @null))
490     {
491         SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
492     }
493     if (!AreEqual(link.Target, @null))
494     {
495         TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
496     }
497     link.Source = this.GetSource(substitution);
498     link.Target = this.GetTarget(substitution);
499     if (!AreEqual(link.Source, @null))
500     {
501         SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
502     }
503     if (!AreEqual(link.Target, @null))
504     {
505         TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
506     }
507     return handler != null ? handler(before, GetLinkStruct(linkIndex)) :
508     ↳ Constants.Continue;
509 }
510
511 /// <remarks>
512 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
513     ↳ пространство
514 /// </remarks>
515 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

511 public virtual TLinkAddress Create(ICollection<TLinkAddress>? substitution,
512     ↳ WriteHandler<TLinkAddress>? handler)
513 {
514     ref var header = ref GetHeaderReference();
515     var freeLink = header.FirstFreeLink;
516     if (!AreEqual(freeLink, Constants.Null))
517     {
518         UnusedLinksListMethods.Detach(freeLink);
519     }
520     else
521     {
522         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
523         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
524         {
525             throw new
526                 ↳ LinksLimitReachedException<TLinkAddress>(maximumPossibleInnerReference);
527         }
528         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
529         {
530             _memory.ReservedCapacity += _memory.ReservationStep;
531             SetPointers(_memory);
532             header = ref GetHeaderReference();
533             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
534                 ↳ LinkSizeInBytes);
535         }
536         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
537         _memory.UsedCapacity += LinkSizeInBytes;
538     }
539     return handler != null ? handler(null, new Link<TLinkAddress>(freeLink,
540         ↳ Constants.Null, Constants.Null)) : Constants.Continue;
541 }
542
543 /// <summary>
544 /// <para>
545 /// Deletes the substitution.
546 /// </para>
547 /// <para></para>
548 /// </summary>
549 /// <param name="restriction">
550 /// <para>The substitution.</para>
551 /// <para></para>
552 /// </param>
553 [MethodImpl(MethodImplOptions.AggressiveInlining)]
554 public virtual TLinkAddress Delete(ICollection<TLinkAddress>? restriction,
555     ↳ WriteHandler<TLinkAddress>? handler)
556 {
557     ref var header = ref GetHeaderReference();
558     var link = restriction[Constants.IndexPart];
559     var before = GetLinkStruct(link);
560     if (LessThan(link, header.AllocatedLinks))
561     {
562         UnusedLinksListMethods.AttachAsFirst(link);
563         // return handler?.Invoke(before, null);
564         return handler != null ? handler(before, null) : Constants.Continue;
565     }
566     else if (AreEqual(link, header.AllocatedLinks))
567     {
568         header.AllocatedLinks = Decrement(header.AllocatedLinks);
569         _memory.UsedCapacity -= LinkSizeInBytes;
570         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
571         // пока не дойдём до первой существующей связи
572         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
573         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
574             ↳ IsUnusedLink(header.AllocatedLinks))
575         {
576             UnusedLinksListMethods.Detach(header.AllocatedLinks);
577             header.AllocatedLinks = Decrement(header.AllocatedLinks);
578             _memory.UsedCapacity -= LinkSizeInBytes;
579         }
580         return handler != null ? handler(before, null) : Constants.Continue;
581     }
582     return Constants.Continue;
583 }
584
585 /// <summary>
586 /// <para>
587 /// Gets the link struct using the specified link index.
588 /// </para>

```

```

582     /// <para></para>
583     /// </summary>
584     /// <param name="linkIndex">
585     /// <para>The link index.</para>
586     /// <para></para>
587     /// </param>
588     /// <returns>
589     /// <para>A list of t link</para>
590     /// <para></para>
591     /// </returns>
592     [MethodImpl(MethodImplOptions.AggressiveInlining)]
593     public IList<TLinkAddress>? GetLinkStruct(TLinkAddress linkIndex)
594     {
595         ref var link = ref GetLinkReference(linkIndex);
596         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
597     }
598
599     /// <remarks>
600     /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
601     /// → адрес реально поменялся
602     /// </remarks>
603     /// Указатель this.links может быть в том же месте,
604     /// так как 0-я связь не используется и имеет такой же размер как Header,
605     /// поэтому header размещается в том же месте, что и 0-я связь
606     /// </remarks>
607     [MethodImpl(MethodImplOptions.AggressiveInlining)]
608     protected abstract void SetPointers(IResizableDirectMemory memory);
609
610     /// <summary>
611     /// <para>
612     /// Resets the pointers.
613     /// </para>
614     /// <para></para>
615     /// </summary>
616     [MethodImpl(MethodImplOptions.AggressiveInlining)]
617     protected virtual void ResetPointers()
618     {
619         SourcesTreeMethods = null;
620         TargetsTreeMethods = null;
621         UnusedLinksListMethods = null;
622     }
623
624     /// <summary>
625     /// <para>
626     /// Gets the header reference.
627     /// </para>
628     /// <para></para>
629     /// </summary>
630     /// <returns>
631     /// <para>A ref links header of t link</para>
632     /// <para></para>
633     /// </returns>
634     [MethodImpl(MethodImplOptions.AggressiveInlining)]
635     protected abstract ref LinksHeader<TLinkAddress> GetHeaderReference();
636
637     /// <summary>
638     /// <para>
639     /// Gets the link reference using the specified link index.
640     /// </para>
641     /// <para></para>
642     /// </summary>
643     /// <param name="linkIndex">
644     /// <para>The link index.</para>
645     /// <para></para>
646     /// </param>
647     /// <returns>
648     /// <para>A ref raw link of t link</para>
649     /// <para></para>
650     /// </returns>
651     [MethodImpl(MethodImplOptions.AggressiveInlining)]
652     protected abstract ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress linkIndex);
653
654     /// <summary>
655     /// <para>
656     /// Determines whether this instance exists.
657     /// </para>
658     /// <para></para>
659     /// </summary>

```

```

659     /// <param name="link">
660     /// <para>The link.</para>
661     /// <para></para>
662     /// </param>
663     /// <returns>
664     /// <para>The bool</para>
665     /// <para></para>
666     /// </returns>
667     [MethodImpl(MethodImplOptions.AggressiveInlining)]
668     protected virtual bool Exists(TLinkAddress link)
669         => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
670             && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
671             && !IsUnusedLink(link);
672
673     /// <summary>
674     /// <para>
675     /// Determines whether this instance is unused link.
676     /// </para>
677     /// <para></para>
678     /// </summary>
679     /// <param name="linkIndex">
680     /// <para>The link index.</para>
681     /// <para></para>
682     /// </param>
683     /// <returns>
684     /// <para>The bool</para>
685     /// <para></para>
686     /// </returns>
687     [MethodImpl(MethodImplOptions.AggressiveInlining)]
688     protected virtual bool IsUnusedLink(TLinkAddress linkIndex)
689     {
690         if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
691             ↳ is not needed
692         {
693             ref var link = ref GetLinkReference(linkIndex);
694             return AreEqual(link.SizeAsSource, default) && !AreEqual(link.Source, default);
695         }
696         else
697         {
698             return true;
699         }
700     }
701
702     /// <summary>
703     /// <para>
704     /// Gets the one.
705     /// </para>
706     /// <para></para>
707     /// </summary>
708     /// <returns>
709     /// <para>The link</para>
710     /// <para></para>
711     /// </returns>
712     [MethodImpl(MethodImplOptions.AggressiveInlining)]
713     protected virtual TLinkAddress GetOne() => _one;
714
715     /// <summary>
716     /// <para>
717     /// Gets the zero.
718     /// </para>
719     /// <para></para>
720     /// </summary>
721     /// <returns>
722     /// <para>The link</para>
723     /// <para></para>
724     /// </returns>
725     [MethodImpl(MethodImplOptions.AggressiveInlining)]
726     protected virtual TLinkAddress GetZero() => default;
727
728     /// <summary>
729     /// <para>
730     /// Determines whether this instance are equal.
731     /// </para>
732     /// <para></para>
733     /// </summary>
734     /// <param name="first">
735     /// <para>The first.</para>
736     /// <para></para>

```

```

736    /// </param>
737    /// <param name="second">
738    /// <para>The second.</para>
739    /// <para></para>
740    /// </param>
741    /// <returns>
742    /// <para>The bool</para>
743    /// <para></para>
744    /// </returns>
745    [MethodImpl(MethodImplOptions.AggressiveInlining)]
746    protected virtual bool AreEqual(TLinkAddress first, TLinkAddress second) =>
747        ↪ _equalityComparer.Equals(first, second);
748
749    /// <summary>
750    /// <para>
751    /// Determines whether this instance less than.
752    /// </para>
753    /// <para></para>
754    /// </summary>
755    /// <param name="first">
756    /// <para>The first.</para>
757    /// <para></para>
758    /// </param>
759    /// <param name="second">
760    /// <para>The second.</para>
761    /// <para></para>
762    /// </param>
763    /// <returns>
764    /// <para>The bool</para>
765    /// <para></para>
766    /// </returns>
767    [MethodImpl(MethodImplOptions.AggressiveInlining)]
768    protected virtual bool LessThan(TLinkAddress first, TLinkAddress second) =>
769        ↪ _comparer.Compare(first, second) < 0;
770
771    /// <summary>
772    /// <para>
773    /// Determines whether this instance less or equal than.
774    /// </para>
775    /// <para></para>
776    /// </summary>
777    /// <param name="first">
778    /// <para>The first.</para>
779    /// <para></para>
780    /// </param>
781    /// <param name="second">
782    /// <para>The second.</para>
783    /// <para></para>
784    /// </param>
785    /// <returns>
786    /// <para>The bool</para>
787    /// <para></para>
788    /// </returns>
789    [MethodImpl(MethodImplOptions.AggressiveInlining)]
790    protected virtual bool LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
791        ↪ _comparer.Compare(first, second) <= 0;
792
793    /// <summary>
794    /// <para>
795    /// Determines whether this instance greater than.
796    /// </para>
797    /// <para></para>
798    /// </summary>
799    /// <param name="first">
800    /// <para>The first.</para>
801    /// <para></para>
802    /// </param>
803    /// <param name="second">
804    /// <para>The second.</para>
805    /// <para></para>
806    /// </param>
807    /// <returns>
808    /// <para>The bool</para>
809    /// <para></para>
810    /// </returns>
811    [MethodImpl(MethodImplOptions.AggressiveInlining)]
812    protected virtual bool GreaterThan(TLinkAddress first, TLinkAddress second) =>
813        ↪ _comparer.Compare(first, second) > 0;

```

```

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

```

```

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.
931     /// </para>
932     /// <para></para>
933     /// </summary>
934     /// <param name="link">
935     /// <para>The link.</para>
936     /// <para></para>
937     /// </param>
938     /// <returns>
939     /// <para>The link</para>
940     /// <para></para>
941     /// </returns>
942     [MethodImpl(MethodImplOptions.AggressiveInlining)]
943     protected virtual TLinkAddress Decrement(TLinkAddress link) =>
944         ↪ Arithmetic<TLinkAddress>.Decrement(link);
945
946     #region Disposable
947
948     /// <summary>
949     /// <para>
950     /// Gets the allow multiple dispose calls value.
951     /// </para>
952     /// <para></para>
953     /// </summary>
954     protected override bool AllowMultipleDisposeCalls
955     {
956         [MethodImpl(MethodImplOptions.AggressiveInlining)]
957         get => true;
958     }
959
960     /// <summary>
961     /// <para>
962     /// Disposes the manual.
963     /// </para>
964     /// </summary>

```

```

959     /// </para>
960     /// <para></para>
961     /// </summary>
962     /// <param name="manual">
963     /// <para>The manual.</para>
964     /// <para></para>
965     /// </param>
966     /// <param name="wasDisposed">
967     /// <para>The was disposed.</para>
968     /// <para></para>
969     /// </param>
970     [MethodImpl(MethodImplOptions.AggressiveInlining)]
971     protected override void Dispose(bool manual, bool wasDisposed)
972     {
973         if (!wasDisposed)
974         {
975             ResetPointers();
976             _memory.DisposeIfPossible();
977         }
978     }
979
980     #endregion
981 }
982 }

```

1.92 ./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> :
19         ↳ AbsoluteCircularDoublyLinkedListMethods<TLinkAddress>, ILinksListMethods<TLinkAddress>
20     {
21         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
22         ↳ = UncheckedConverter<TLinkAddress, long>.Default;
23         private readonly byte* _links;
24         private readonly byte* _header;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="links">
33         /// <para>A links.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UnusedLinksListMethods(byte* links, byte* header)
42         {
43             _links = links;
44             _header = header;
45         }
46
47         /// <summary>
48         /// <para>
49         /// Gets the header reference.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <returns>

```

```

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) =>
        ↳ GetLinkReference(element).Source;

117
118     /// <summary>
119     /// <para>
120     /// Gets the next using the specified element.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="element">
125     /// <para>The element.</para>

```



```

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

```

```

200     /// </param>
201     /// <param name="next">
202     /// <para>The next.</para>
203     /// <para></para>
204     /// </param>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override void SetNext(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     /// <para></para>
214     /// </summary>
215     /// <param name="size">
216     /// <para>The size.</para>
217     /// <para></para>
218     /// </param>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override void SetSize(TLinkAddress size) => GetHeaderReference().FreeLinks =
221         ↪ size;
222 }

```

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

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.United
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLink<TLinkAddress> : IEquatable<RawLink<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<RawLink<TLinkAddress>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLinkAddress Source;
36
37         /// <summary>
38         /// <para>
39         /// The target.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLinkAddress Target;
44
45         /// <summary>
46         /// <para>
47         /// The left as source.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public TLinkAddress LeftAsSource;
52
53         /// <summary>
54         /// <para>
55         /// The right as source.
56         /// </para>
57         /// </summary>
58         public TLinkAddress RightAsSource;
59     }
60 }

```

```

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)
        ↪ => left.Equals(right);
142
143     [MethodImpl(MethodImplOptions.AggressiveInlining)]
144     public static bool operator !=(RawLink<TLinkAddress> left, RawLink<TLinkAddress> right)
        ↪ => !(left == right);
145 }
146 }

```

1.94 ./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>
        ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header)
52         : base(constants, (byte*)links, (byte*)header)
53     {
54         Links = links;
55     }
56 }

```

```

54         Header = header;
55     }
56
57     /// <summary>
58     /// <para>
59     /// Gets the zero.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <returns>
64     /// <para>The uint</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override uint GetZero() => 0U;
69
70     /// <summary>
71     /// <para>
72     /// Determines whether this instance equal to zero.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="value">
77     /// <para>The value.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The bool</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override bool EqualToZero(uint value) => value == 0U;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance are equal.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="first">
94     /// <para>The first.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="second">
98     /// <para>The second.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The bool</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override bool AreEqual(uint first, uint second) => first == second;
107
108    /// <summary>
109    /// <para>
110    /// Determines whether this instance greater than zero.
111    /// </para>
112    /// <para></para>
113    /// </summary>
114    /// <param name="value">
115    /// <para>The value.</para>
116    /// <para></para>
117    /// </param>
118    /// <returns>
119    /// <para>The bool</para>
120    /// <para></para>
121    /// </returns>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    protected override bool GreaterThanZero(uint value) => value > 0U;
124
125    /// <summary>
126    /// <para>
127    /// Determines whether this instance greater than.
128    /// </para>
129    /// <para></para>
130    /// </summary>
131    /// <param name="first">

```

```

132    /// <para>The first.</para>
133    /// <para></para>
134    /// </param>
135    /// <param name="second">
136    /// <para>The second.</para>
137    /// <para></para>
138    /// </param>
139    /// <returns>
140    /// <para>The bool</para>
141    /// <para></para>
142    /// </returns>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override bool GreaterThan(uint first, uint second) => first > second;
145
146    /// <summary>
147    /// <para>
148    /// Determines whether this instance greater or equal than.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="first">
153    /// <para>The first.</para>
154    /// <para></para>
155    /// </param>
156    /// <param name="second">
157    /// <para>The second.</para>
158    /// <para></para>
159    /// </param>
160    /// <returns>
161    /// <para>The bool</para>
162    /// <para></para>
163    /// </returns>
164    [MethodImpl(MethodImplOptions.AggressiveInlining)]
165    protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
166
167    /// <summary>
168    /// <para>
169    /// Determines whether this instance greater or equal than zero.
170    /// </para>
171    /// <para></para>
172    /// </summary>
173    /// <param name="value">
174    /// <para>The value.</para>
175    /// <para></para>
176    /// </param>
177    /// <returns>
178    /// <para>The bool</para>
179    /// <para></para>
180    /// </returns>
181    [MethodImpl(MethodImplOptions.AggressiveInlining)]
182    protected override bool GreaterOrEqualThanZero(uint value) => true; // value >= 0 is
    ↪ 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     /// </summary>
266     /// <param name="value">
267     /// <para>The value.</para>
268     /// <para></para>
269     /// </param>
270     /// <returns>
271     /// <para>The uint</para>
272     /// <para></para>
273     /// </returns>
274     [MethodImpl(MethodImplOptions.AggressiveInlining)]
275     protected override uint Increment(uint value) => ++value;
276
277     /// <summary>
278     /// <para>
279     /// Decrements the value.
280     /// </para>
281     /// <para></para>
282     /// </summary>
283     /// <param name="value">
284     /// <para>The value.</para>

```

```

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

```



```

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

```

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

```

```

18    /// <para>
19    /// The links.
20    /// </para>
21    /// <para></para>
22    /// </summary>
23    protected new readonly RawLink<uint>* Links;
24    /// <summary>
25    /// <para>
26    /// The header.
27    /// </para>
28    /// <para></para>
29    /// </summary>
30    protected new readonly LinksHeader<uint>* Header;
31
32    /// <summary>
33    /// <para>
34    /// Initializes a new <see cref="UInt32LinksSizeBalancedTreeMethodsBase"/> instance.
35    /// </para>
36    /// <para></para>
37    /// </summary>
38    /// <param name="constants">
39    /// <para>A constants.</para>
40    /// <para></para>
41    /// </param>
42    /// <param name="links">
43    /// <para>A links.</para>
44    /// <para></para>
45    /// </param>
46    /// <param name="header">
47    /// <para>A header.</para>
48    /// <para></para>
49    /// </param>
50    protected UInt32LinksSizeBalancedTreeMethodsBase(LinksConstants<uint> constants,
51    ↪ RawLink<uint>* links, LinksHeader<uint>* header)
52    : base(constants, (byte*)links, (byte*)header)
53    {
54        Links = links;
55        Header = header;
56    }
57    /// <summary>
58    /// <para>
59    /// Gets the zero.
60    /// </para>
61    /// <para></para>
62    /// </summary>
63    /// <returns>
64    /// <para>The uint</para>
65    /// <para></para>
66    /// </returns>
67    [MethodImpl(MethodImplOptions.AggressiveInlining)]
68    protected override uint GetZero() => 0U;
69
70    /// <summary>
71    /// <para>
72    /// Determines whether this instance equal to zero.
73    /// </para>
74    /// <para></para>
75    /// </summary>
76    /// <param name="value">
77    /// <para>The value.</para>
78    /// <para></para>
79    /// </param>
80    /// <returns>
81    /// <para>The bool</para>
82    /// <para></para>
83    /// </returns>
84    [MethodImpl(MethodImplOptions.AggressiveInlining)]
85    protected override bool EqualToZero(uint value) => value == 0U;
86
87    /// <summary>
88    /// <para>
89    /// Determines whether this instance are equal.
90    /// </para>
91    /// <para></para>
92    /// </summary>
93    /// <param name="first">
94    /// <para>The first.</para>

```

```

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

```

```

173     /// <param name="value">
174     /// <para>The value.</para>
175     /// <para></para>
176     /// </param>
177     /// <returns>
178     /// <para>The bool</para>
179     /// <para></para>
180     /// </returns>
181     [MethodImpl(MethodImplOptions.AggressiveInlining)]
182     protected override bool GreaterOrEqualThanZero(uint value) => true; // value >= 0 is
    ↪ 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     /// <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 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links sources recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods :
15         ↳ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↳ cref="UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// <para></para>
35         /// </param>
36         public UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
37             ↳ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
38             ↳ header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="node">
47         /// <para>The node.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The ref uint</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.

```

```

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>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void ClearNode(uint node)
278     {
279         ref var link = ref Links[node];
280         link.LeftAsSource = 0U;
281         link.RightAsSource = 0U;
282         link.SizeAsSource = 0U;
283     }
284 }
285 }

```

1.97 ./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>
47        /// </param>
48        /// <returns>
49        /// <para>The ref uint</para>
50        /// <para></para>
51        /// </returns>
52        [MethodImpl(MethodImplOptions.AggressiveInlining)]
53        protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
54
55        /// <summary>
56        /// <para>
57        /// Gets the right reference using the specified node.
58        /// </para>
59        /// <para></para>
60        /// </summary>
61        /// <param name="node">
62        /// <para>The node.</para>
63        /// <para></para>
64        /// </param>
65        /// <returns>
66        /// <para>The ref uint</para>
67        /// <para></para>
68        /// </returns>
69        [MethodImpl(MethodImplOptions.AggressiveInlining)]
70        protected override ref uint GetRightReference(uint node) => ref
71            ↳ Links[node].RightAsSource;
72
73        /// <summary>
74        /// <para>
75        /// Gets the left using the specified node.
76        /// </para>
77        /// <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 FirstIsToTheLeftOfSecond(uint firstSource, uint firstTarget,
230     ↪ uint secondSource, uint secondTarget)
231     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
232     ↪ secondTarget);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance first is to the right of second.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="firstSource">
241     /// <para>The first source.</para>
242     /// <para></para>
243     /// </param>
244     /// <param name="firstTarget">
245     /// <para>The first target.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="secondSource">
249     /// <para>The second source.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="secondTarget">
253     /// <para>The second target.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>
257     /// <para>The bool</para>
258     /// <para></para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
262     ↪ uint secondSource, uint secondTarget)
263     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
264     ↪ secondTarget);
265
266     /// <summary>
267     /// <para>
268     /// Clears the node using the specified node.
269     /// </para>
270     /// <para></para>
271     /// </summary>
272     /// <param name="node">
273     /// <para>The node.</para>
274     /// <para></para>
275     /// </param>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void ClearNode(uint node)
278     {
279         ref var link = ref Links[node];
280         link.LeftAsSource = 0U;
281         link.RightAsSource = 0U;
282         link.SizeAsSource = 0U;
283     }
284 }

```

1.98 ./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>

```

```

18      /// Initializes a new <see
19      ↪ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20      /// </para>
21      /// </summary>
22      /// <param name="constants">
23      /// <para>A constants.</para>
24      /// </para>
25      /// </param>
26      /// <param name="links">
27      /// <para>A links.</para>
28      /// </para>
29      /// </param>
30      /// <param name="header">
31      /// <para>A header.</para>
32      /// </para>
33      /// </param>
34      public UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
35      ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
36      ↪ header) { }
37
38      /// <summary>
39      /// <para>
40      /// Gets the left reference using the specified node.
41      /// </para>
42      /// </summary>
43      /// <param name="node">
44      /// <para>The node.</para>
45      /// </para>
46      /// </param>
47      /// <returns>
48      /// <para>The ref uint</para>
49      /// </returns>
50      [MethodImpl(MethodImplOptions.AggressiveInlining)]
51      protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsTarget;
52
53      /// <summary>
54      /// <para>
55      /// Gets the right reference using the specified node.
56      /// </para>
57      /// </summary>
58      /// <param name="node">
59      /// <para>The node.</para>
60      /// </para>
61      /// </param>
62      /// <returns>
63      /// <para>The ref uint</para>
64      /// </returns>
65      [MethodImpl(MethodImplOptions.AggressiveInlining)]
66      protected override ref uint GetRightReference(uint node) => ref
67      ↪ Links[node].RightAsTarget;
68
69      /// <summary>
70      /// <para>
71      /// Gets the left using the specified node.
72      /// </para>
73      /// </summary>
74      /// <param name="node">
75      /// <para>The node.</para>
76      /// </para>
77      /// </param>
78      /// <returns>
79      /// <para>The uint</para>
80      /// </returns>
81      [MethodImpl(MethodImplOptions.AggressiveInlining)]
82      protected override uint GetLeft(uint node) => Links[node].LeftAsTarget;
83
84      /// <summary>
85      /// <para>
86      /// Gets the right using the specified node.
87      /// </para>
88      /// </summary>
89      /// <para>
90      /// </para>
91

```

```

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

```



```

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

```

```

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 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksTargetsSizeBalancedTreeMethods :
        ↪ UInt32LinksSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="constants">
23         /// <para>A constants.</para>
24         /// <para></para>
25         /// </param>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>
32         /// <para></para>
33         /// </param>
34         public UInt32LinksTargetsSizeBalancedTreeMethods(LinksConstants<uint> constants,
        ↪ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
35

```

```

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

```

```

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

```

```

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 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Memory;
4  using Platform.Singletons;
5  using Platform.Data.Doublets.Memory.United.Generic;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     ///     ↪ organizing the storage of links with addresses represented as <see cref="uint" />.</para>
14     /// <para>Представляет низкоуровневую реализацию прямого доступа к памяти с переменным
15     ///     ↪ размером, для организации хранения связей с адресами представленными в виде <see
16     ///     ↪ cref="uint"/>.</para>
17     /// </summary>
18     public unsafe class UInt32UnitedMemoryLinks : UnitedMemoryLinksBase<uint>
19     {
20         private readonly Func<ILinksTreeMethods<uint>> _createSourceTreeMethods;
21         private readonly Func<ILinksTreeMethods<uint>> _createTargetTreeMethods;
22         private LinksHeader<uint>* _header;
23         private RawLink<uint>* _links;
24
25         /// <summary>
26         /// <para>
27         ///     Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="address">
32         /// <para>A address.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt32UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
37
38         /// <summary>
39         /// <para>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 UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
47         ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
48         ↪ memoryReservationStep) { }
49
50         /// <summary>
51         /// <para>
52         ///     Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <param name="memory">
57         /// <para>A memory.</para>
58         /// <para></para>
59         /// </param>

```

```

52     /// </param>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
55         ↪ DefaultLinksSizeStep) { }
56
57     /// <summary>
58     /// <para>
59     ///     Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
60     /// </para>
61     /// </summary>
62     /// <param name="memory">
63     ///     <para>A memory.</para>
64     /// </para>
65     /// </param>
66     /// <param name="memoryReservationStep">
67     ///     <para>A memory reservation step.</para>
68     /// </para>
69     /// </param>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
72         ↪ memoryReservationStep) : this(memory, memoryReservationStep,
73         ↪ Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }
74
75     /// <summary>
76     /// <para>
77     ///     Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
78     /// </para>
79     /// </summary>
80     /// <param name="memory">
81     ///     <para>A memory.</para>
82     /// </para>
83     /// </param>
84     /// <param name="memoryReservationStep">
85     ///     <para>A memory reservation step.</para>
86     /// </para>
87     /// </param>
88     /// <param name="constants">
89     ///     <para>A constants.</para>
90     /// </para>
91     /// </param>
92     /// <param name="indexTreeType">
93     ///     <para>A index tree type.</para>
94     /// </para>
95     /// </param>
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
98         ↪ memoryReservationStep, LinksConstants<uint> constants, IndexTreeType indexTreeType)
99         ↪ : base(memory, memoryReservationStep, constants)
100     {
101         if (indexTreeType == IndexTreeType.SizeBalancedTree)
102         {
103             _createSourceTreeMethods = () => new
104                 ↪ UInt32LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
105             _createTargetTreeMethods = () => new
106                 ↪ UInt32LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
107         }
108         else
109         {
110             _createSourceTreeMethods = () => new
111                 ↪ UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
112                 ↪ _header);
113             _createTargetTreeMethods = () => new
114                 ↪ UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
115                 ↪ _header);
116         }
117         Init(memory, memoryReservationStep);
118     }
119
120     /// <summary>
121     /// <para>
122     ///     Sets the pointers using the specified memory.
123     /// </para>
124     /// </summary>
125     /// <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
    ↪ _links[linkIndex];
174
175 /// <summary>
176 /// <para>
177 /// Determines whether this instance are equal.
178 /// </para>
179 /// <para></para>
180 /// </summary>
181 /// <param name="first">
182 /// <para>The first.</para>
183 /// <para></para>
184 /// </param>
185 /// <param name="second">
186 /// <para>The second.</para>
187 /// <para></para>
188 /// </param>
189 /// <returns>
190 /// <para>The bool</para>
191 /// <para></para>
192 /// </returns>
193 [MethodImpl(MethodImplOptions.AggressiveInlining)]
194 protected override bool AreEqual(uint first, uint second) => first == second;

```



```

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

```

```

273     /// <returns>
274     /// <para>The bool</para>
275     /// <para></para>
276     /// </returns>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
279
280     /// <summary>
281     /// <para>
282     /// Gets the zero.
283     /// </para>
284     /// <para></para>
285     /// </summary>
286     /// <returns>
287     /// <para>The uint</para>
288     /// <para></para>
289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     protected override uint GetZero() => 0U;
292
293     /// <summary>
294     /// <para>
295     /// Gets the one.
296     /// </para>
297     /// <para></para>
298     /// </summary>
299     /// <returns>
300     /// <para>The uint</para>
301     /// <para></para>
302     /// </returns>
303     [MethodImpl(MethodImplOptions.AggressiveInlining)]
304     protected override uint GetOne() => 1U;
305
306     /// <summary>
307     /// <para>
308     /// Converts the to int 64 using the specified value.
309     /// </para>
310     /// <para></para>
311     /// </summary>
312     /// <param name="value">
313     /// <para>The value.</para>
314     /// <para></para>
315     /// </param>
316     /// <returns>
317     /// <para>The long</para>
318     /// <para></para>
319     /// </returns>
320     [MethodImpl(MethodImplOptions.AggressiveInlining)]
321     protected override long ConvertToInt64(uint value) => (long)value;
322
323     /// <summary>
324     /// <para>
325     /// Converts the to address using the specified value.
326     /// </para>
327     /// <para></para>
328     /// </summary>
329     /// <param name="value">
330     /// <para>The value.</para>
331     /// <para></para>
332     /// </param>
333     /// <returns>
334     /// <para>The uint</para>
335     /// <para></para>
336     /// </returns>
337     [MethodImpl(MethodImplOptions.AggressiveInlining)]
338     protected override uint ConvertToAddress(long value) => (uint)value;
339
340     /// <summary>
341     /// <para>
342     /// Adds the first.
343     /// </para>
344     /// <para></para>
345     /// </summary>
346     /// <param name="first">
347     /// <para>The first.</para>
348     /// <para></para>
349     /// </param>
350     /// <param name="second">

```

```

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

```

1.101 ./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 }

```

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

```

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

```

```

16 public unsafe abstract class UInt64LinksAvlBalancedTreeMethodsBase :
    ↳ LinksAvlBalancedTreeMethodsBase<ulong>
17 {
18     /// <summary>
19     /// <para>
20     /// The links.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     protected new readonly RawLink<ulong>* Links;
25     /// <summary>
26     /// <para>
27     /// The header.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     protected new readonly LinksHeader<ulong>* Header;
32
33     /// <summary>
34     /// <para>
35     /// Initializes a new <see cref="UInt64LinksAvlBalancedTreeMethodsBase"/> instance.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     /// <param name="constants">
40     /// <para>A constants.</para>
41     /// <para></para>
42     /// </param>
43     /// <param name="links">
44     /// <para>A links.</para>
45     /// <para></para>
46     /// </param>
47     /// <param name="header">
48     /// <para>A header.</para>
49     /// <para></para>
50     /// </param>
51     protected UInt64LinksAvlBalancedTreeMethodsBase(LinksConstants<ulong> constants,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header)
52     : base(constants, (byte*)links, (byte*)header)
53     {
54         Links = links;
55         Header = header;
56     }
57
58     /// <summary>
59     /// <para>
60     /// Gets the zero.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <returns>
65     /// <para>The ulong</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ulong GetZero() => 0UL;
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 == 0UL;
87
88     /// <summary>
89     /// <para>
90     /// Determines whether this instance are equal.
91     /// </para>

```

```

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

```

```

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

```

```

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

```



```

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

```

```

399     /// <returns>
400     /// <para>The ulong</para>
401     /// <para></para>
402     /// </returns>
403     [MethodImpl(MethodImplOptions.AggressiveInlining)]
404     protected override ulong GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;
405
406     /// <summary>
407     /// <para>
408     /// Sets the size value using the specified stored value.
409     /// </para>
410     /// <para></para>
411     /// </summary>
412     /// <param name="storedValue">
413     /// <para>The stored value.</para>
414     /// <para></para>
415     /// </param>
416     /// <param name="size">
417     /// <para>The size.</para>
418     /// <para></para>
419     /// </param>
420     [MethodImpl(MethodImplOptions.AggressiveInlining)]
421     protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
422         ↪ storedValue & 31UL | (size & 134217727UL) << 5;
423
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance get left is child value.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="value">
431     /// <para>The value.</para>
432     /// <para></para>
433     /// </param>
434     /// <returns>
435     /// <para>The bool</para>
436     /// <para></para>
437     /// </returns>
438     [MethodImpl(MethodImplOptions.AggressiveInlining)]
439     protected override bool GetLeftIsChildValue(ulong value) => (value & 16UL) >> 4 == 1UL;
440
441     /// <summary>
442     /// <para>
443     /// Sets the left is child value using the specified stored value.
444     /// </para>
445     /// <para></para>
446     /// </summary>
447     /// <param name="storedValue">
448     /// <para>The stored value.</para>
449     /// <para></para>
450     /// </param>
451     /// <param name="value">
452     /// <para>The value.</para>
453     /// <para></para>
454     /// </param>
455     [MethodImpl(MethodImplOptions.AggressiveInlining)]
456     protected override void SetLeftIsChildValue(ref ulong storedValue, bool value) =>
457         ↪ storedValue = storedValue & 4294967279UL | (As<bool, byte>(ref value) & 1UL) << 4;
458
459     /// <summary>
460     /// <para>
461     /// Determines whether this instance get right is child value.
462     /// </para>
463     /// <para></para>
464     /// </summary>
465     /// <param name="value">
466     /// <para>The value.</para>
467     /// <para></para>
468     /// </param>
469     /// <returns>
470     /// <para>The bool</para>
471     /// <para></para>
472     /// </returns>
473     [MethodImpl(MethodImplOptions.AggressiveInlining)]
474     protected override bool GetRightIsChildValue(ulong value) => (value & 8UL) >> 3 == 1UL;
475
476     /// <summary>

```

```

475     /// <para>
476     /// Sets the right is child value using the specified stored value.
477     /// </para>
478     /// <para></para>
479     /// </summary>
480     /// <param name="storedValue">
481     /// <para>The stored value.</para>
482     /// <para></para>
483     /// </param>
484     /// <param name="value">
485     /// <para>The value.</para>
486     /// <para></para>
487     /// </param>
488     [MethodImpl(MethodImplOptions.AggressiveInlining)]
489     protected override void SetRightIsChildValue(ref ulong storedValue, bool value) =>
490         ↪ storedValue = storedValue & 4294967287UL | (As<bool, byte>(ref value) & 1UL) << 3;
491
492     /// <summary>
493     /// <para>
494     /// Gets the balance value using the specified value.
495     /// </para>
496     /// <para></para>
497     /// </summary>
498     /// <param name="value">
499     /// <para>The value.</para>
500     /// <para></para>
501     /// </param>
502     /// <returns>
503     /// <para>The sbyte</para>
504     /// <para></para>
505     /// </returns>
506     [MethodImpl(MethodImplOptions.AggressiveInlining)]
507     protected override sbyte GetBalanceValue(ulong value) => unchecked((sbyte)(value & 7UL |
508         ↪ 0xF8UL * ((value & 4UL) >> 2))); // if negative, then continue ones to the end of
509         ↪ sbyte
510
511     /// <summary>
512     /// <para>
513     /// Sets the balance value using the specified stored value.
514     /// </para>
515     /// <para></para>
516     /// </summary>
517     /// <param name="storedValue">
518     /// <para>The stored value.</para>
519     /// <para></para>
520     /// </param>
521     /// <param name="value">
522     /// <para>The value.</para>
523     /// <para></para>
524     /// </param>
525     [MethodImpl(MethodImplOptions.AggressiveInlining)]
526     protected override void SetBalanceValue(ref ulong storedValue, sbyte value) =>
527         ↪ storedValue = unchecked(storedValue & 4294967288UL | (ulong)((byte)value >> 5 & 4 |
528         ↪ value & 3) & 7UL);
529
530     /// <summary>
531     /// <para>
532     /// Gets the header reference.
533     /// </para>
534     /// <para></para>
535     /// </summary>
536     /// <returns>
537     /// <para>A ref links header of ulong</para>
538     /// <para></para>
539     /// </returns>
540     [MethodImpl(MethodImplOptions.AggressiveInlining)]
541     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
542
543     /// <summary>
544     /// <para>
545     /// Gets the link reference using the specified link.
546     /// </para>
547     /// <para></para>
548     /// </summary>
549     /// <param name="link">
550     /// <para>The link.</para>
551     /// <para></para>
552     /// </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 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10         /// Represents the int 64 links recursionless size balanced tree methods base.
11         /// </para>
12         /// <para></para>
13         /// </summary>
14         /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{ulong}" />
15         public unsafe abstract class UInt64LinksRecursionlessSizeBalancedTreeMethodsBase :
16             ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<ulong>
17         {
18             /// <summary>
19             /// <para>
20             /// The links.
21             /// </para>
22             /// <para></para>
23             /// </summary>
24             protected new readonly RawLink<ulong>* Links;
25             /// <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="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase" />
36             ↳ instance.
37             /// </para>
38             /// <para></para>
39             /// </summary>
40             /// <param name="constants">
41             /// <para>A constants.</para>
42             /// <para></para>
43             /// </param>
44             /// <param name="links">
45             /// <para>A links.</para>
46             /// <para></para>
47             /// </param>
48             /// <param name="header">
49             /// <para>A header.</para>
50             /// <para></para>
51             /// </param>
52             protected UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
53             ↳ constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
54             : base(constants, (byte*)links, (byte*)header)
55         {
56             Links = links;
57             Header = header;
58         }
59
60         /// <summary>
61         /// <para>
62         /// Gets the zero.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <returns>
67         /// <para>The ulong</para>
68         /// <para></para>
69         /// </returns>

```

```

67 [MethodImpl(MethodImplOptions.AggressiveInlining)]
68 protected override ulong GetZero() => OUL;
69
70 /// <summary>
71 /// <para>
72 /// Determines whether this instance equal to zero.
73 /// </para>
74 /// <para></para>
75 /// </summary>
76 /// <param name="value">
77 /// <para>The value.</para>
78 /// <para></para>
79 /// </param>
80 /// <returns>
81 /// <para>The bool</para>
82 /// <para></para>
83 /// </returns>
84 [MethodImpl(MethodImplOptions.AggressiveInlining)]
85 protected override bool EqualToZero(ulong value) => value == OUL;
86
87 /// <summary>
88 /// <para>
89 /// Determines whether this instance are equal.
90 /// </para>
91 /// <para></para>
92 /// </summary>
93 /// <param name="first">
94 /// <para>The first.</para>
95 /// <para></para>
96 /// </param>
97 /// <param name="second">
98 /// <para>The second.</para>
99 /// <para></para>
100 /// </param>
101 /// <returns>
102 /// <para>The bool</para>
103 /// <para></para>
104 /// </returns>
105 [MethodImpl(MethodImplOptions.AggressiveInlining)]
106 protected override bool AreEqual(ulong first, ulong second) => first == second;
107
108 /// <summary>
109 /// <para>
110 /// Determines whether this instance greater than zero.
111 /// </para>
112 /// <para></para>
113 /// </summary>
114 /// <param name="value">
115 /// <para>The value.</para>
116 /// <para></para>
117 /// </param>
118 /// <returns>
119 /// <para>The bool</para>
120 /// <para></para>
121 /// </returns>
122 [MethodImpl(MethodImplOptions.AggressiveInlining)]
123 protected override bool GreaterThanZero(ulong value) => value > OUL;
124
125 /// <summary>
126 /// <para>
127 /// Determines whether this instance greater than.
128 /// </para>
129 /// <para></para>
130 /// </summary>
131 /// <param name="first">
132 /// <para>The first.</para>
133 /// <para></para>
134 /// </param>
135 /// <param name="second">
136 /// <para>The second.</para>
137 /// <para></para>
138 /// </param>
139 /// <returns>
140 /// <para>The bool</para>
141 /// <para></para>
142 /// </returns>
143 [MethodImpl(MethodImplOptions.AggressiveInlining)]
144 protected override bool GreaterThan(ulong first, ulong second) => first > second;

```

```

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

```

```

221     /// <summary>
222     /// <para>
223     /// Determines whether this instance less than zero.
224     /// </para>
225     /// <para></para>
226     /// </summary>
227     /// <param name="value">
228     /// <para>The value.</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 LessThanZero(ulong value) => false; // value < 0 is always false
237     ↪ 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>

```



```

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 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 links size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksSizeBalancedTreeMethodsBase{ulong}"/>
15     public unsafe abstract class UInt64LinksSizeBalancedTreeMethodsBase :
16     ↪ LinksSizeBalancedTreeMethodsBase<ulong>
17     {
18         /// <summary>
19         /// <para>
20         /// The links.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         protected new readonly RawLink<ulong>* Links;
25
26         /// <summary>
27         /// <para>
28         /// The header.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly LinksHeader<ulong>* Header;

```

```

31
32     /// <summary>
33     /// <para>
34     /// Initializes a new <see cref="UInt64LinksSizeBalancedTreeMethodsBase"/> instance.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="constants">
39     /// <para>A constants.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="links">
43     /// <para>A links.</para>
44     /// <para></para>
45     /// </param>
46     /// <param name="header">
47     /// <para>A header.</para>
48     /// <para></para>
49     /// </param>
50     protected UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
51     ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
52     : base(constants, (byte*)links, (byte*)header)
53     {
54         Links = links;
55         Header = header;
56     }
57     /// <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 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links sources avl balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksSourcesAvlBalancedTreeMethods :
15         ↳ UInt64LinksAvlBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64LinksSourcesAvlBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
36             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37             ↳ { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="node">
46         /// <para>The node.</para>
47         /// <para></para>
48         /// </param>
49         /// <returns>
50         /// <para>The ref ulong</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref ulong GetLeftReference(ulong node) => ref
55             ↳ Links[node].LeftAsSource;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref ulong</para>
69         /// <para></para>
70         /// </returns>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```



```

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

```

```

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

```

```

/// <summary>
/// <para>
/// Gets the base part value using the specified link.
/// </para>
/// </summary>
/// <param name="link">
/// <para>The link.</para>
/// </param>
/// <returns>
/// <para>The ulong</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>
/// </summary>
/// <param name="firstSource">
/// <para>The first source.</para>
/// </param>
/// <param name="firstTarget">
/// <para>The first target.</para>
/// </param>
/// <param name="secondSource">
/// <para>The second source.</para>
/// </param>
/// <param name="secondTarget">
/// <para>The second target.</para>
/// </param>
/// <returns>
/// <para>The bool</para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
    ↪ ulong secondSource, ulong secondTarget)
    => firstSource < secondSource || (firstSource == secondSource && firstTarget <
    ↪ secondTarget);

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

```

```

367     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
368         ↪ ulong secondSource, ulong secondTarget)
369         => firstSource > secondSource || (firstSource == secondSource && firstTarget >
370             ↪ secondTarget);
371
372     /// <summary>
373     /// <para>
374     /// Clears the node using the specified node.
375     /// </para>
376     /// </summary>
377     /// <param name="node">
378     /// <para>The node.</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 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links sources recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods :
15         ↪ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// <para></para>
35         /// </param>
36         public UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
37             ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
38             ↪ links, header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="node">
47         /// <para>The node.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <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>
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].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(ulong firstSource, ulong firstTarget,
230     ↪ ulong secondSource, ulong secondTarget)
231     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
232     ↪ secondTarget);
233
234 /// <summary>
235 /// <para>
236 /// Determines whether this instance first is to the right of second.
237 /// </para>
238 /// <para></para>
239 /// </summary>
240 /// <param name="firstSource">
241 /// <para>The first source.</para>
242 /// <para></para>
243 /// </param>
244 /// <param name="firstTarget">
245 /// <para>The first target.</para>
246 /// <para></para>
247 /// </param>
248 /// <param name="secondSource">
249 /// <para>The second source.</para>
250 /// <para></para>
251 /// </param>
252 /// <param name="secondTarget">
253 /// <para>The second target.</para>
254 /// <para></para>
255 /// </param>
256 /// <returns>
257 /// <para>The bool</para>
258 /// <para></para>
259 /// </returns>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
262     ↪ ulong secondSource, ulong secondTarget)
263     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
264     ↪ secondTarget);
265
266 /// <summary>
267 /// <para>
268 /// Clears the node using the specified node.
269 /// </para>
270 /// <para></para>
271 /// </summary>
272 /// <param name="node">
273 /// <para>The node.</para>
274 /// <para></para>
275 /// </param>

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links sources size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksSourcesSizeBalancedTreeMethods :
15         ↪ UInt64LinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64LinksSourcesSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
36             ↪ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37             ↪ { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="node">
46         /// <para>The node.</para>
47         /// <para></para>
48         /// </param>
49         /// <returns>
50         /// <para>The ref ulong</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref ulong GetLeftReference(ulong node) => ref
55             ↪ Links[node].LeftAsSource;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>

```



```

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>

```

```

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

```

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

```

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

```

```

6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 64 links targets avl balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
14    public unsafe class UInt64LinksTargetsAvlBalancedTreeMethods :
15        ↳ UInt64LinksAvlBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt64LinksTargetsAvlBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="header">
32        /// <para>A header.</para>
33        /// <para></para>
34        /// </param>
35        public UInt64LinksTargetsAvlBalancedTreeMethods(LinksConstants<ulong> constants,
36            ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37        { }
38
39        /// <summary>
40        /// <para>
41        /// Gets the left reference using the specified node.
42        /// </para>
43        /// <para></para>
44        /// </summary>
45        /// <param name="node">
46        /// <para>The node.</para>
47        /// <para></para>
48        /// </param>
49        /// <returns>
50        /// <para>The ref ulong</para>
51        /// <para></para>
52        /// </returns>
53        [MethodImpl(MethodImplOptions.AggressiveInlining)]
54        protected override ref ulong GetLeftReference(ulong node) => ref
55            ↳ Links[node].LeftAsTarget;
56
57        /// <summary>
58        /// <para>
59        /// Gets the right reference using the specified node.
60        /// </para>
61        /// <para></para>
62        /// </summary>
63        /// <param name="node">
64        /// <para>The node.</para>
65        /// <para></para>
66        /// </param>
67        /// <returns>
68        /// <para>The ref ulong</para>
69        /// <para></para>
70        /// </returns>
71        [MethodImpl(MethodImplOptions.AggressiveInlining)]
72        protected override ref ulong GetRightReference(ulong node) => ref
73            ↳ Links[node].RightAsTarget;
74
75        /// <summary>
76        /// <para>
77        /// Gets the left using the specified node.
78        /// </para>
79        /// <para></para>
80        /// </summary>
81        /// <param name="node">
82        /// <para>The node.</para>
83        /// <para></para>
84        /// </param>
85        /// <returns>
86        /// <para>The ref ulong</para>
87        /// <para></para>
88        /// </returns>
89        [MethodImpl(MethodImplOptions.AggressiveInlining)]
90        protected override ref ulong GetLeftReferenceUsingNode(ulong node) => ref
91            ↳ Links[node].LeftUsingNode;
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 ref ulong</para>
105       /// <para></para>
106       /// </returns>
107       [MethodImpl(MethodImplOptions.AggressiveInlining)]
108       protected override ref ulong GetRightReferenceUsingNode(ulong node) => ref
109           ↳ Links[node].RightUsingNode;
110    }
111 }

```

```

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

```

```

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

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

188
189     /// <summary>
190     /// <para>
191     /// Sets the left is child using the specified node.
192     /// </para>
193     /// <para></para>
194     /// </summary>
195     /// <param name="node">
196     /// <para>The node.</para>
197     /// <para></para>
198     /// </param>
199     /// <param name="value">
200     /// <para>The value.</para>
201     /// <para></para>
202     /// </param>
203     [MethodImpl(MethodImplOptions.AggressiveInlining)]
204     protected override void SetLeftIsChild(ulong node, bool value) =>
        ↳ SetLeftIsChildValue(ref Links[node].SizeAsTarget, value);

205
206     /// <summary>
207     /// <para>
208     /// Determines whether this instance get right is child.
209     /// </para>
210     /// <para></para>
211     /// </summary>
212     /// <param name="node">
213     /// <para>The node.</para>
214     /// <para></para>
215     /// </param>
216     /// <returns>
217     /// <para>The bool</para>
218     /// <para></para>
219     /// </returns>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     protected override bool GetRightIsChild(ulong node) =>
        ↳ GetRightIsChildValue(Links[node].SizeAsTarget);

222
223     /// <summary>
224     /// <para>
225     /// Sets the right is child using the specified node.
226     /// </para>
227     /// <para></para>
228     /// </summary>

```

```

229     /// <param name="node">
230     /// <para>The node.</para>
231     /// <para></para>
232     /// </param>
233     /// <param name="value">
234     /// <para>The value.</para>
235     /// <para></para>
236     /// </param>
237     [MethodImpl(MethodImplOptions.AggressiveInlining)]
238     protected override void SetRightIsChild(ulong node, bool value) =>
239         ↪ SetRightIsChildValue(ref Links[node].SizeAsTarget, value);
240
241     /// <summary>
242     /// <para>
243     /// Gets the balance using the specified node.
244     /// </para>
245     /// <para></para>
246     /// </summary>
247     /// <param name="node">
248     /// <para>The node.</para>
249     /// <para></para>
250     /// </param>
251     /// <returns>
252     /// <para>The sbyte</para>
253     /// <para></para>
254     /// </returns>
255     [MethodImpl(MethodImplOptions.AggressiveInlining)]
256     protected override sbyte GetBalance(ulong node) =>
257         ↪ GetBalanceValue(Links[node].SizeAsTarget);
258
259     /// <summary>
260     /// <para>
261     /// Sets the balance using the specified node.
262     /// </para>
263     /// <para></para>
264     /// </summary>
265     /// <param name="node">
266     /// <para>The node.</para>
267     /// <para></para>
268     /// </param>
269     /// <param name="value">
270     /// <para>The value.</para>
271     /// <para></para>
272     /// </param>
273     [MethodImpl(MethodImplOptions.AggressiveInlining)]
274     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
275         ↪ Links[node].SizeAsTarget, value);
276
277     /// <summary>
278     /// <para>
279     /// Gets the tree root.
280     /// </para>
281     /// <para></para>
282     /// </summary>
283     /// <returns>
284     /// <para>The ulong</para>
285     /// <para></para>
286     /// </returns>
287     [MethodImpl(MethodImplOptions.AggressiveInlining)]
288     protected override ulong GetTreeRoot() => Header->RootAsTarget;
289
290     /// <summary>
291     /// <para>
292     /// Gets the base part value using the specified link.
293     /// </para>
294     /// <para></para>
295     /// </summary>
296     /// <param name="link">
297     /// <para>The link.</para>
298     /// <para></para>
299     /// </param>
300     /// <returns>
301     /// <para>The ulong</para>
302     /// <para></para>
303     /// </returns>
304     [MethodImpl(MethodImplOptions.AggressiveInlining)]
305     protected override ulong GetBasePartValue(ulong link) => Links[link].Target;

```

```

304     /// <summary>
305     /// <para>
306     /// Determines whether this instance first is to the left of second.
307     /// </para>
308     /// <para></para>
309     /// </summary>
310     /// <param name="firstSource">
311     /// <para>The first source.</para>
312     /// <para></para>
313     /// </param>
314     /// <param name="firstTarget">
315     /// <para>The first target.</para>
316     /// <para></para>
317     /// </param>
318     /// <param name="secondSource">
319     /// <para>The second source.</para>
320     /// <para></para>
321     /// </param>
322     /// <param name="secondTarget">
323     /// <para>The second target.</para>
324     /// <para></para>
325     /// </param>
326     /// <returns>
327     /// <para>The bool</para>
328     /// <para></para>
329     /// </returns>
330     [MethodImpl(MethodImplOptions.AggressiveInlining)]
331     protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
332         ↪ ulong secondSource, ulong secondTarget)
333         => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
334             ↪ secondSource);
335
336     /// <summary>
337     /// <para>
338     /// Determines whether this instance first is to the right of second.
339     /// </para>
340     /// <para></para>
341     /// </summary>
342     /// <param name="firstSource">
343     /// <para>The first source.</para>
344     /// <para></para>
345     /// </param>
346     /// <param name="firstTarget">
347     /// <para>The first target.</para>
348     /// <para></para>
349     /// </param>
350     /// <param name="secondSource">
351     /// <para>The second source.</para>
352     /// <para></para>
353     /// </param>
354     /// <param name="secondTarget">
355     /// <para>The second target.</para>
356     /// <para></para>
357     /// </param>
358     /// <returns>
359     /// <para>The bool</para>
360     /// <para></para>
361     /// </returns>
362     [MethodImpl(MethodImplOptions.AggressiveInlining)]
363     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
364         ↪ ulong secondSource, ulong secondTarget)
365         => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
366             ↪ secondSource);
367
368     /// <summary>
369     /// <para>
370     /// Clears the node using the specified node.
371     /// </para>
372     /// <para></para>
373     /// </summary>
374     /// <param name="node">
375     /// <para>The node.</para>
376     /// <para></para>
377     /// </param>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     protected override void ClearNode(ulong node)
380     {

```



```

377         ref var link = ref Links[node];
378         link.LeftAsTarget = OUL;
379         link.RightAsTarget = OUL;
380         link.SizeAsTarget = OUL;
381     }
382 }
383 }

```

1.109 ./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="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods :
15         ↳ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↳ cref="UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// <para></para>
35         /// </param>
36         public UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
37         ↳ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
38         ↳ links, header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="node">
47         /// <para>The node.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The ref ulong</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref ulong GetLeftReference(ulong node) => ref
56         ↳ Links[node].LeftAsTarget;
57
58         /// <summary>
59         /// <para>
60         /// Gets the right reference using the specified node.
61         /// </para>
62         /// <para></para>
63         /// </summary>
64         /// <param name="node">
65         /// <para>The node.</para>
66         /// <para></para>
67         /// </param>
68         /// <returns>
69         /// <para>The ref ulong</para>
70         /// <para></para>
71         /// </returns>

```

```

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

```

1.110 ./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 :
    ↳ UInt64LinksSizeBalancedTreeMethodsBase
15 {
16     /// <summary>
17     /// <para>
18     /// Initializes a new <see cref="UInt64LinksTargetsSizeBalancedTreeMethods"/> 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 UInt64LinksTargetsSizeBalancedTreeMethods(LinksConstants<ulong> constants,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
    ↳ { }
35
36     /// <summary>
37     /// <para>
38     /// Gets the left reference using the specified node.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref ulong</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
    ↳ Links[node].LeftAsTarget;
52
53     /// <summary>
54     /// <para>
55     /// Gets the right reference using the specified node.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
    ↳ Links[node].RightAsTarget;
69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>

```

```

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

```

```

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,
        ↳ ulong secondSource, ulong secondTarget)
230         => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
        ↳ secondSource);

```

```

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

```

1.111 `./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 }

```



```

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

```

```

92  /// <para>A index tree type.</para>
93  /// <para></para>
94  /// </param>
95  [MethodImpl(MethodImplOptions.AggressiveInlining)]
96  public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
    ↳ memoryReservationStep, LinksConstants<ulong> constants, IndexTreeType indexTreeType)
    ↳ : base(memory, memoryReservationStep, constants)
97  {
98      if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
99      {
100         _createSourceTreeMethods = () => new
            ↳ UInt64LinksSourcesAvlBalancedTreeMethods(Constants, _links, _header);
101         _createTargetTreeMethods = () => new
            ↳ UInt64LinksTargetsAvlBalancedTreeMethods(Constants, _links, _header);
102     }
103     else if (indexTreeType == IndexTreeType.SizeBalancedTree)
104     {
105         _createSourceTreeMethods = () => new
            ↳ UInt64LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
106         _createTargetTreeMethods = () => new
            ↳ UInt64LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
107     }
108     else
109     {
110         _createSourceTreeMethods = () => new
            ↳ UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
            ↳ _header);
111         _createTargetTreeMethods = () => new
            ↳ UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
            ↳ _header);
112     }
113     Init(memory, memoryReservationStep);
114 }
115
116 /// <summary>
117 /// <para>
118 /// Sets the pointers using the specified memory.
119 /// </para>
120 /// <para></para>
121 /// </summary>
122 /// <param name="memory">
123 /// <para>The memory.</para>
124 /// <para></para>
125 /// </param>
126 [MethodImpl(MethodImplOptions.AggressiveInlining)]
127 protected override void SetPointers(IResizableDirectMemory memory)
128 {
129     _header = (LinksHeader<ulong>*)memory.Pointer;
130     _links = (RawLink<ulong>*)memory.Pointer;
131     SourcesTreeMethods = _createSourceTreeMethods();
132     TargetsTreeMethods = _createTargetTreeMethods();
133     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
134 }
135
136 /// <summary>
137 /// <para>
138 /// Resets the pointers.
139 /// </para>
140 /// <para></para>
141 /// </summary>
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 protected override void ResetPointers()
144 {
145     base.ResetPointers();
146     _links = null;
147     _header = null;
148 }
149
150 /// <summary>
151 /// <para>
152 /// Gets the header reference.
153 /// </para>
154 /// <para></para>
155 /// </summary>
156 /// <returns>
157 /// <para>A ref links header of ulong</para>
158 /// <para></para>

```

```

159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
162
163     /// <summary>
164     /// <para>
165     /// Gets the link reference using the specified link index.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="linkIndex">
170     /// <para>The link index.</para>
171     /// <para></para>
172     /// </param>
173     /// <returns>
174     /// <para>A ref raw link of ulong</para>
175     /// <para></para>
176     /// </returns>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
179     ↪ _links[linkIndex];
180
181     /// <summary>
182     /// <para>
183     /// Determines whether this instance are equal.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="first">
188     /// <para>The first.</para>
189     /// <para></para>
190     /// </param>
191     /// <param name="second">
192     /// <para>The second.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The bool</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override bool AreEqual(ulong first, ulong second) => first == second;
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance less than.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="first">
209     /// <para>The first.</para>
210     /// <para></para>
211     /// </param>
212     /// <param name="second">
213     /// <para>The second.</para>
214     /// <para></para>
215     /// </param>
216     /// <returns>
217     /// <para>The bool</para>
218     /// <para></para>
219     /// </returns>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     protected override bool LessThan(ulong first, ulong second) => first < second;
222
223     /// <summary>
224     /// <para>
225     /// Determines whether this instance less or equal than.
226     /// </para>
227     /// <para></para>
228     /// </summary>
229     /// <param name="first">
230     /// <para>The first.</para>
231     /// <para></para>
232     /// </param>
233     /// <param name="second">
234     /// <para>The second.</para>
235     /// <para></para>
236     /// </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 }

```

1.112 ./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 }

```

1.113 ./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>
19     {
20         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
21         ↪ EqualityComparer<TLinkAddress>.Default;
22
23         /// <summary>
24         /// <para>
25         /// Initializes a new <see cref="PropertiesOperator"/> instance.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         /// <param name="links">
30         /// <para>A links.</para>
31         /// <para></para>
32         /// </param>
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public PropertiesOperator(ILinks<TLinkAddress> links) : base(links) { }
35
36         /// <summary>
37         /// <para>
38         /// Gets the value using the specified object.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="@object">
43         /// <para>The object.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="property">
47         /// <para>The property.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The link</para>
52         /// <para></para>

```

```

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(links.GetIndex(valueLink));
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 }

```

1.114 ./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>
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>

```



```

27     /// <para></para>
28     /// </summary>
29     /// <param name="links">
30     /// <para>A links.</para>
31     /// <para></para>
32     /// </param>
33     /// <param name="propertyMarker">
34     /// <para>A property marker.</para>
35     /// <para></para>
36     /// </param>
37     /// <param name="propertyValueMarker">
38     /// <para>A property value marker.</para>
39     /// <para></para>
40     /// </param>
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     public PropertyOperator(ILinks<TLinkAddress> links, TLinkAddress propertyMarker,
43     ↪ TLinkAddress propertyValueMarker) : base(links)
44     {
45         _propertyMarker = propertyMarker;
46         _propertyValueMarker = propertyValueMarker;
47     }
48     /// <summary>
49     /// <para>
50     /// Gets the link.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     /// <param name="link">
55     /// <para>The link.</para>
56     /// <para></para>
57     /// </param>
58     /// <returns>
59     /// <para>The link</para>
60     /// <para></para>
61     /// </returns>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public TLinkAddress Get(TLinkAddress link)
64     {
65         var property = _links.SearchOrDefault(link, _propertyMarker);
66         return GetValue(GetContainer(property));
67     }
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     private TLinkAddress GetContainer(TLinkAddress property)
70     {
71         var valueContainer = default(TLinkAddress);
72         if (_equalityComparer.Equals(property, default))
73         {
74             return valueContainer;
75         }
76         var links = _links;
77         var constants = links.Constants;
78         var countinueConstant = constants.Continue;
79         var breakConstant = constants.Break;
80         var anyConstant = constants.Any;
81         var query = new Link<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     /// <summary>
99     /// <para>
100     /// Sets the link.
101     /// </para>
102     /// <para></para>

```

```

103     /// </summary>
104     /// <param name="link">
105     /// <para>The link.</para>
106     /// <para></para>
107     /// </param>
108     /// <param name="value">
109     /// <para>The value.</para>
110     /// <para></para>
111     /// </param>
112     [MethodImpl(MethodImplOptions.AggressiveInlining)]
113     public void Set(TLinkAddress link, TLinkAddress value)
114     {
115         var links = _links;
116         var property = links.GetOrCreate(link, _propertyMarker);
117         var container = GetContainer(property);
118         if (_equalityComparer.Equals(container, default))
119         {
120             links.GetOrCreate(property, value);
121         }
122         else
123         {
124             links.Update(container, property, value);
125         }
126     }
127 }
128 }

```

1.115 ./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         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
21             ↪ EqualityComparer<TLinkAddress>.Default;
22         private readonly TLinkAddress _stack;
23
24         /// <summary>
25         /// <para>
26         /// Gets the is empty value.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         public bool IsEmpty
31         {
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             get => _equalityComparer.Equals(Peek(), _stack);
34         }
35
36         /// <summary>
37         /// <para>
38         /// Initializes a new <see cref="Stack" /> instance.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="links">
43         /// <para>A links.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="stack">
47         /// <para>A stack.</para>
48         /// <para></para>
49         /// </param>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

50     public Stack(ILinks<TLinkAddress> links, TLinkAddress stack) : base(links) => _stack =
    ↪     stack;
51     [MethodImpl(MethodImplOptions.AggressiveInlining)]
52     private TLinkAddress GetStackMarker() => _links.GetSource(_stack);
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     private TLinkAddress GetTop() => _links.GetTarget(_stack);
55
56     /// <summary>
57     /// <para>
58     ///     Peeks this instance.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <returns>
63     ///     <para>The link</para>
64     /// <para></para>
65     /// </returns>
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
67     public TLinkAddress Peek() => _links.GetTarget(GetTop());
68
69     /// <summary>
70     /// <para>
71     ///     Pops this instance.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <returns>
76     ///     <para>The element.</para>
77     /// <para></para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public TLinkAddress Pop()
81     {
82         var element = Peek();
83         if (!_equalityComparer.Equals(element, _stack))
84         {
85             var top = GetTop();
86             var previousTop = _links.GetSource(top);
87             _links.Update(_stack, GetStackMarker(), previousTop);
88             _links.Delete(top);
89         }
90         return element;
91     }
92
93     /// <summary>
94     /// <para>
95     ///     Pushes the element.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="element">
100    /// <para>The element.</para>
101    /// <para></para>
102    /// </param>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    public void Push(TLinkAddress element) => _links.Update(_stack, GetStackMarker(),
    ↪     _links.GetOrCreate(GetTop(), element));
105 }
106 }

```

1.116 ./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,
39     ↪ TLinkAddress stackMarker)
40     {
41         var stackPoint = links.CreatePoint();
42         var stack = links.Update(stackPoint, stackMarker, stackPoint);
43         return stack;
44     }
45 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Data.Doublets;
5  using Platform.Delegates;
6  using Platform.Threading.Synchronization;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets
11 {
12     /// <remarks>
13     /// TODO: Autogeneration of synchronized wrapper (decorator).
14     /// TODO: Try to unfold code of each method using IL generation for performance improvements.
15     /// TODO: Or even to unfold multiple layers of implementations.
16     /// </remarks>
17     public class SynchronizedLinks<TLinkAddress> : ISynchronizedLinks<TLinkAddress>
18     {
19         /// <summary>
20         /// <para>
21         /// Gets the constants value.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public LinksConstants<TLinkAddress> Constants
26         {
27             [MethodImpl(MethodImplOptions.AggressiveInlining)]
28             get;
29         }
30
31         /// <summary>
32         /// <para>
33         /// Gets the sync root value.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         public ISynchronization SyncRoot
38         {
39             [MethodImpl(MethodImplOptions.AggressiveInlining)]
40             get;
41         }
42
43         /// <summary>
44         /// <para>
45         /// Gets the sync value.
46         /// </para>
47         /// <para></para>
48         /// </summary>

```

```

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

```

```

125     /// </summary>
126     /// <param name="handler">
127     /// <para>The handler.</para>
128     /// <para></para>
129     /// </param>
130     /// <param name="restriction">
131     /// <para>The substitution.</para>
132     /// <para></para>
133     /// </param>
134     /// <returns>
135     /// <para>The link address</para>
136     /// <para></para>
137     /// </returns>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     public TLinkAddress Each(ICollection<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
        ↳ handler) => SyncRoot.DoRead(restriction, handler, Unsync.Each);
140
141     /// <summary>
142     /// <para>
143     /// Creates the substitution.
144     /// </para>
145     /// <para></para>
146     /// </summary>
147     /// <param name="substitution">
148     /// <para>The substitution.</para>
149     /// <para></para>
150     /// </param>
151     /// <returns>
152     /// <para>The link address</para>
153     /// <para></para>
154     /// </returns>
155     [MethodImpl(MethodImplOptions.AggressiveInlining)]
156     public TLinkAddress Create(ICollection<TLinkAddress>? substitution,
        ↳ WriteHandler<TLinkAddress>? handler) => SyncRoot.DoWrite(substitution, handler,
        ↳ Unsync.Create);
157
158     /// <summary>
159     /// <para>
160     /// Updates the substitution.
161     /// </para>
162     /// <para></para>
163     /// </summary>
164     /// <param name="restriction">
165     /// <para>The substitution.</para>
166     /// <para></para>
167     /// </param>
168     /// <param name="substitution">
169     /// <para>The substitution.</para>
170     /// <para></para>
171     /// </param>
172     /// <returns>
173     /// <para>The link address</para>
174     /// <para></para>
175     /// </returns>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     public TLinkAddress Update(ICollection<TLinkAddress>? restriction, ICollection<TLinkAddress>?
        ↳ substitution, WriteHandler<TLinkAddress>? handler) => SyncRoot.DoWrite(restriction,
        ↳ substitution, handler, Unsync.Update);
178
179     /// <summary>
180     /// <para>
181     /// Deletes the substitution.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="restriction">
186     /// <para>The substitution.</para>
187     /// <para></para>
188     /// </param>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     public TLinkAddress Delete(ICollection<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
        ↳ handler) => SyncRoot.DoWrite(restriction, handler, Unsync.Delete);
191
192     //public T Trigger(ICollection<T> restriction, Func<ICollection<T>, ICollection<T>, T> matchedHandler,
        ↳ ICollection<T> substitution, Func<ICollection<T>, ICollection<T>, T> substitutedHandler)
193     //{

```

```

194         // if (restriction != null && substitution != null &&
195         ↪ !substitution.EqualTo(restriction))
196         // return SyncRoot.ExecuteWriteOperation(restriction, matchedHandler,
197         ↪ substitution, substitutedHandler, Unsync.Trigger);
198
199         // return SyncRoot.ExecuteReadOperation(restriction, matchedHandler, substitution,
200         ↪ substitutedHandler, Unsync.Trigger);
201     //}
202 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Singletons;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the int 64 links extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class UInt64LinksExtensions
18     {
19         /// <summary>
20         /// <para>
21         /// The instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public static readonly LinksConstants<ulong> Constants =
26             ↪ Default<LinksConstants<ulong>>.Instance;
27
28         /// <summary>
29         /// <para>
30         /// Determines whether any link is any.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="links">
35         /// <para>The links.</para>
36         /// <para></para>
37         /// </param>
38         /// <param name="sequence">
39         /// <para>The sequence.</para>
40         /// <para></para>
41         /// </param>
42         /// <returns>
43         /// <para>The bool</para>
44         /// <para></para>
45         /// </returns>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static bool AnyLinkIsAny(this ILinks<ulong> links, params ulong[] sequence)
48         {
49             if (sequence == null)
50             {
51                 return false;
52             }
53             var constants = links.Constants;
54             for (var i = 0; i < sequence.Length; i++)
55             {
56                 if (sequence[i] == constants.Any)
57                 {
58                     return true;
59                 }
60             }
61             return false;
62         }
63
64         /// <summary>
65         /// <para>
66         /// Formats the structure using the specified links.
67         /// </para>

```

```

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>
117     /// <para></para>
118     /// </param>
119     /// <param name="isElement">
120     /// <para>The is element.</para>
121     /// <para></para>
122     /// </param>
123     /// <param name="appendElement">
124     /// <para>The append element.</para>
125     /// <para></para>
126     /// </param>
127     /// <param name="renderIndex">
128     /// <para>The render index.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="renderDebug">
132     /// <para>The render debug.</para>
133     /// <para></para>
134     /// </param>
135     /// <returns>
136     /// <para>The string</para>
137     /// <para></para>
138     /// </returns>
139     [MethodImpl(MethodImplOptions.AggressiveInlining)]
140     public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
141     ↪ Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
142     ↪ bool renderIndex = false, bool renderDebug = false)
143     {

```



```

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,
223                 ↪ appendElement, renderIndex);
224         }
225     }
226     sb.Append(' ');
227     if (link.Target == link.Index)
228     {
229         sb.Append(link.Index);
230     }
231     else
232     {
233         var target = new Link<ulong>(links.GetLink(link.Target));
234         if (isElement(target))
235         {
236             appendElement(sb, target);
237         }
238         else
239         {
240             links.AppendStructure(sb, visited, target.Index, isElement,
241                 ↪ appendElement, renderIndex);
242         }
243     }
244     sb.Append(')');
245 }
246 else
247 {
248     if (renderDebug)
249     {
250         sb.Append('*');
251     }
252     sb.Append(linkIndex);
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }

```

1.119 ./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     /// <remarks>
31     /// Альтернативные варианты хранения трансформации (элемента транзакции):
32     ///
33     /// private enum TransitionType
34     /// {
35     ///     Creation,
36     ///     UpdateOf,
37     ///     UpdateTo,
38     ///     Deletion
39     /// }
40     ///
41     /// private struct Transition
42     /// {
43     ///     public TLinkAddress TransactionId;
44     ///     public UniqueTimestamp Timestamp;
45     ///     public TransactionItemType Type;
46     ///     public Link Source;
47     ///     public Link Linker;
48     ///     public Link Target;
49     /// }
50     ///
51     /// Или
52     ///
53     /// public struct TransitionHeader
54     /// {
55     ///     public TLinkAddress TransactionIdCombined;
56     ///     public TLinkAddress TimestampCombined;
57     ///
58     ///     public TLinkAddress TransactionId
59     ///     {
60     ///         get
61     ///         {
62     ///             return (TLinkAddress) mask & TransactionIdCombined;
63     ///         }
64     ///     }
65     ///
66     ///     public UniqueTimestamp Timestamp
67     ///     {
68     ///         get
69     ///         {
70     ///             return (UniqueTimestamp)mask & TransactionIdCombined;
71     ///         }
72     ///     }
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     /// <summary>
187     /// <para>
188     /// Initializes a new <see cref="Transition"/> instance.
189     /// </para>
190     /// <para></para>
191     /// </summary>
192     /// <param name="uniqueTimestampFactory">
193     /// <para>A unique timestamp factory.</para>
194     /// <para></para>
195     /// </param>
196     /// <param name="transactionId">
197     /// <para>A transaction id.</para>
198     /// <para></para>
199     /// </param>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     public Transition(UniqueTimestampFactory uniqueTimestampFactory, TLinkAddress
        ↳ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
        ↳ }

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

214     /// <summary>
215     /// <para>
216     /// Determines whether this instance equals.
217     /// </para>
218     /// <para></para>
219     /// </summary>
220     /// <param name="obj">
221     /// <para>The obj.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     public override bool Equals(object obj) => obj is Transition transition ?
        ↳ Equals(transition) : false;

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

```

```

246
247     /// <summary>
248     /// <para>
249     /// Determines whether this instance equals.
250     /// </para>
251     /// </summary>
252     /// <param name="other">
253     /// <para>The other.</para>
254     /// </param>
255     /// <returns>
256     /// <para>The bool</para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     public bool Equals(Transition other) => TransactionId == other.TransactionId &&
260     ↪ Before == other.Before && After == other.After && Timestamp == other.Timestamp;
261
262     [MethodImpl(MethodImplOptions.AggressiveInlining)]
263     public static bool operator ==(Transition left, Transition right) =>
264     ↪ left.Equals(right);
265
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     public static bool operator !=(Transition left, Transition right) => !(left ==
268     ↪ 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.
293
294 /// На жёстком диске:
295 /// Минусы:
296 /// 1. Длительный отклик, на запись каждой трансформации.
297 /// 2. Лог транзакций дополнительно наполняется отменёнными транзакциями.
298 /// -> Это может решаться упаковкой/исключением дублирующих операций.
299 /// -> Также это может решаться тем, что короткие транзакции вообще
300 ↪ не будут записываться в случае отката.
301 /// 3. Перед тем как выполнять отмену операций транзакции нужно дождаться пока все
302 ↪ операции (трансформации)
303 ↪ будут записаны в лог.
304
305 /// </remarks>
306 public class Transaction : DisposableBase
307 {
308     private readonly Queue<Transition> _transitions;
309     private readonly UInt64LinksTransactionsLayer _layer;
310     /// <summary>
311     /// <para>
312     /// Gets or sets the is committed value.
313     /// </para>
314     /// </summary>
315     public bool IsCommitted { get; private set; }
316     /// <summary>
317     /// <para>
318     /// Gets or sets the is reverted value.
319     /// </para>
320     /// </summary>

```

```

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     /// <para></para>
383     /// </summary>
384     /// <param name="layer">
385     /// <para>The layer.</para>
386     /// <para></para>
387     /// </param>
388     /// <param name="transaction">
389     /// <para>The transaction.</para>
390     /// <para></para>
391     /// </param>
392     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

393 public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
394 ↪ Transaction transaction)
395 {
396     layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
397     layer._currentTransactionTransitions = transaction._transitions;
398     layer._currentTransaction = transaction;
399 }
400
401 /// <summary>
402 /// <para>
403 /// Ensures the transaction allows write operations using the specified transaction.
404 /// </para>
405 /// </summary>
406 /// <param name="transaction">
407 /// <para>The transaction.</para>
408 /// </param>
409 /// <exception cref="InvalidOperationException">
410 /// <para>Transation is committed.</para>
411 /// </exception>
412 /// <exception cref="InvalidOperationException">
413 /// <para>Transation is reverted.</para>
414 /// </exception>
415 [MethodImpl(MethodImplOptions.AggressiveInlining)]
416 public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
417 {
418     if (transaction.IsReverted)
419     {
420         throw new InvalidOperationException("Transation is reverted.");
421     }
422     if (transaction.IsCommitted)
423     {
424         throw new InvalidOperationException("Transation is committed.");
425     }
426 }
427
428 /// <summary>
429 /// <para>
430 /// Disposes the manual.
431 /// </para>
432 /// </summary>
433 /// <param name="manual">
434 /// <para>The manual.</para>
435 /// </param>
436 /// <param name="wasDisposed">
437 /// <para>The was disposed.</para>
438 /// </param>
439 [MethodImpl(MethodImplOptions.AggressiveInlining)]
440 protected override void Dispose(bool manual, bool wasDisposed)
441 {
442     if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
443     {
444         if (!IsCommitted && !IsReverted)
445         {
446             Revert();
447         }
448         _layer.ResetCurrentTransation();
449     }
450 }
451
452 /// <summary>
453 /// <para>
454 /// The from seconds.
455 /// </para>
456 /// </summary>
457 public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
458 private readonly string _logAddress;
459 private readonly FileStream _log;
460 private readonly Queue<Transition> _transitions;
461 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)
520     {
521         FileHelpers.WriteFirst(logAddress, lastCommittedTransition);
522     }
523     _lastCommittedTransition = lastCommittedTransition;
524     // TODO: Think about a better way to calculate or store this value
525     var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
526     _lastCommittedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
527     ↳ x.TransactionId) : 0;
528     _uniqueTimestampFactory = new UniqueTimestampFactory();
529     _logAddress = logAddress;
530     _log = FileHelpers.Append(logAddress);
531     _transitions = new Queue<Transition>();
532     _transitionsPusher = new Task(TransitionsPusher);
533     _transitionsPusher.Start();
534 }
535
536 /// <summary>
537 /// <para>
538 /// Gets the link value using the specified link.
539 /// </para>
540 /// <para></para>
541 /// </summary>
542 /// <param name="link">
543 /// <para>The link.</para>
544 /// <para></para>
545 /// </param>
546 /// <returns>
547 /// <para>A list of TLinkAddress</para>

```

```

546 /// <para></para>
547 /// </returns>
548 [MethodImpl(MethodImplOptions.AggressiveInlining)]
549 public IList<TLinkAddress> GetLinkValue(TLinkAddress link) => _links.GetLink(link);
550
551 /// <summary>
552 /// <para>
553 /// Creates the substitution.
554 /// </para>
555 /// <para></para>
556 /// </summary>
557 /// <param name="substitution">
558 /// <para>The substitution.</para>
559 /// <para></para>
560 /// </param>
561 /// <returns>
562 /// <para>The created link index.</para>
563 /// <para></para>
564 /// </returns>
565 [MethodImpl(MethodImplOptions.AggressiveInlining)]
566 public override TLinkAddress Create(IList<TLinkAddress>? substitution,
    ↳ WriteHandler<TLinkAddress>? handler)
567 {
568     return _links.Create(new Link<TLinkAddress>(), (before, after) =>
569     {
570         CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
571         ↳ new Link<TLinkAddress>(before), new Link<TLinkAddress>(after)));
572         return handler?.Invoke(before, after) ?? Links.Constants.Continue;
573     });
574 }
575
576 /// <summary>
577 /// <para>
578 /// Updates the substitution.
579 /// </para>
580 /// <para></para>
581 /// </summary>
582 /// <param name="restriction">
583 /// <para>The substitution.</para>
584 /// <para></para>
585 /// </param>
586 /// <param name="substitution">
587 /// <para>The substitution.</para>
588 /// <para></para>
589 /// </param>
590 /// <returns>
591 /// <para>The link index.</para>
592 /// <para></para>
593 /// </returns>
594 [MethodImpl(MethodImplOptions.AggressiveInlining)]
595 public override TLinkAddress Update(IList<TLinkAddress>? restriction,
    ↳ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
596 {
597     return _links.Update(restriction, substitution, (before, after) =>
598     {
599         CommitTransition(new Transition(_uniqueTimestampFactory,
600         ↳ _currentTransactionId, new Link<TLinkAddress>(before), new
601         ↳ Link<TLinkAddress>(after)));
602         return handler != null ? handler(before, after) : Constants.Continue;
603     });
604 }
605
606 /// <summary>
607 /// <para>
608 /// Deletes the substitution.
609 /// </para>
610 /// <para></para>
611 /// </summary>
612 /// <param name="restriction">
613 /// <para>The substitution.</para>
614 /// <para></para>
615 /// </param>
616 [MethodImpl(MethodImplOptions.AggressiveInlining)]
617 public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
    ↳ WriteHandler<TLinkAddress>? handler)
618 {

```

```

617     var link = this.GetIndex(restriction);
618     return _links.Delete(restriction, (before, after) =>
619     {
620         CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
621             ↪ before, after));
622         return handler != null ? handler(before, after) : Constants.Continue;
623     });
624 [MethodImpl(MethodImplOptions.AggressiveInlining)]
625 private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
626     ↪ _transitions;
627 [MethodImpl(MethodImplOptions.AggressiveInlining)]
628 private void CommitTransition(Transition transition)
629 {
630     if (_currentTransaction != null)
631     {
632         Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
633     }
634     var transitions = GetCurrentTransitions();
635     transitions.Enqueue(transition);
636 }
637 [MethodImpl(MethodImplOptions.AggressiveInlining)]
638 private void RevertTransition(Transition transition)
639 {
640     if (transition.After.IsNull()) // Revert Deletion with Creation
641     {
642         _links.Create();
643     }
644     else if (transition.Before.IsNull()) // Revert Creation with Deletion
645     {
646         _links.Delete(transition.After.Index);
647     }
648     else // Revert Update
649     {
650         _links.Update(new[] { transition.After.Index, transition.Before.Source,
651             ↪ transition.Before.Target });
652     }
653 }
654 [MethodImpl(MethodImplOptions.AggressiveInlining)]
655 private void ResetCurrentTransaction()
656 {
657     _currentTransactionId = 0;
658     _currentTransactionTransitions = null;
659     _currentTransaction = null;
660 }
661 [MethodImpl(MethodImplOptions.AggressiveInlining)]
662 private void PushTransitions()
663 {
664     if (_log == null || _transitions == null)
665     {
666         return;
667     }
668     for (var i = 0; i < _transitions.Count; i++)
669     {
670         var transition = _transitions.Dequeue();
671         _log.Write(transition);
672         _lastCommittedTransition = transition;
673     }
674 }
675 [MethodImpl(MethodImplOptions.AggressiveInlining)]
676 private void TransitionsPusher()
677 {
678     while (!Disposable.IsDisposed && _transitionsPusher != null)
679     {
680         Thread.Sleep(DefaultPushDelay);
681         PushTransitions();
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 }

```

1.120 ./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
        ↳ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
        ↳ implementation of tree cuts out 5 bits from the address space.
36         Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
        ↳ stMultipleRandomCreationsAndDeletions(100));
37         Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
        ↳ MultipleRandomCreationsAndDeletions(100));
38         Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
        ↳ tMultipleRandomCreationsAndDeletions(100));
39     }
40     private static void Using<TLinkAddress>(Action<ILinks<TLinkAddress>> action)
41     {
42         var unitedMemoryLinks = new UnitedMemoryLinks<TLinkAddress>(new
        ↳ HeapResizableDirectMemory());
43         using (var logFile = File.Open("linksLogger.txt", FileMode.Create, FileAccess.Write))
44         {
45             LoggingDecorator<TLinkAddress> links = new(unitedMemoryLinks, logFile);
46             action(links);
47         }
48
49         File.Delete("db.links");
50         using var ffiLinks = new FFI.UnitedMemoryLinks<TLinkAddress>("db.links");
51         action(ffiLinks);
52     }
53 }
54 }

```

1.121 ./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 }

```

1.122 ./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      }

```

1.123 ./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             [Fact]
25             public static void BasicHeapMemoryTest()
26             {
27                 using (var memory = new
28                     ↪ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
29                 using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
30                     ↪ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
31                 {
32                     memoryAdapter.TestBasicMemoryOperations();
33                 }
34             }
35
36             private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
37             {
38                 var link = memoryAdapter.Create();

```

```

36     memoryAdapter.Delete(link);
37 }
38
39 [Fact]
40 public static void NonexistentReferencesHeapMemoryTest()
41 {
42     using (var memory = new
43         ↪ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
44     using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
45         ↪ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
46     {
47         memoryAdapter.TestNonexistentReferences();
48     }
49 }
50 private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
51 {
52     var link = memoryAdapter.Create();
53     memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);
54     var resultLink = _constants.Null;
55     memoryAdapter.Each(foundLink =>
56     {
57         resultLink = memoryAdapter.GetIndex(foundLink);
58         return _constants.Break;
59     }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
60     Assert.True(resultLink == link);
61     Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
62     memoryAdapter.Delete(link);
63 }

```

1.124 ./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,
        ↳ UnitedMemoryLinks<ulong>>>())
50         {
51             var links = scope.Use<ILinks<ulong>>>();
52             Assert.IsType<UnitedMemoryLinks<ulong>>>(links);
53         }
54     }
55 }
56 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Generic;
5  using Platform.Data.Doublets.Memory;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryGenericLinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using<byte>(links => links.TestCRUDOperations());
15             Using<ushort>(links => links.TestCRUDOperations());
16             Using<uint>(links => links.TestCRUDOperations());
17             Using<ulong>(links => links.TestCRUDOperations());
18         }
19
20         [Fact]
21         public static void RawNumbersCRUDTest()
22         {
23             UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
24             UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
25             UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
26             UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
27         }
28
29         [Fact]
30         public static void MultipleRandomCreationsAndDeletionsTest()
31         {
32             Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
        ↳ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
        ↳ implementation of tree cuts out 5 bits from the address space.
33             Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
        ↳ stMultipleRandomCreationsAndDeletions(100));
34             Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
        ↳ MultipleRandomCreationsAndDeletions(100));
35             Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
        ↳ tMultipleRandomCreationsAndDeletions(100));
36         }
37         private static void Using<TLinkAddress>(Action<ILinks<TLinkAddress>> action)
38         {
39             using (var dataMemory = new HeapResizableDirectMemory())
40             using (var indexMemory = new HeapResizableDirectMemory())
41             using (var memory = new SplitMemoryLinks<TLinkAddress>(dataMemory, indexMemory))
42             {
43                 action(memory);
44             }
45         }
46         private static void
        ↳ UsingWithExternalReferences<TLinkAddress>(Action<ILinks<TLinkAddress>> action)
47         {
48             var constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
        ↳ true);
49             using (var dataMemory = new HeapResizableDirectMemory())
50             using (var indexMemory = new HeapResizableDirectMemory())
51             using (var memory = new SplitMemoryLinks<TLinkAddress>(dataMemory, indexMemory,
        ↳ SplitMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, constants))
52             {
53                 action(memory);
54             }
55         }
56     }
57 }

```


1.126 ./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     }
50 }

```

1.127 ./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().TestMultip
        ↪ leRandomCreationsAndDeletions(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:
        ↪ true);
40         using (var dataMemory = new HeapResizableDirectMemory())
41         using (var indexMemory = new HeapResizableDirectMemory())
42         using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
        ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, constants))
43         {
44             action(memory);
45         }
46     }
47 }
48 }

```

1.128 ./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  using Platform.Delegates;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     public static class TestExtensions
13     {
14         public static void TestCRUDOperations<T>(this ILinks<T> links)
15         {
16             var constants = links.Constants;
17
18             var equalityComparer = EqualityComparer<T>.Default;
19
20             var zero = default(T);
21             var one = Arithmetic.Increment(zero);
22
23             // Create Link
24             Assert.True(equalityComparer.Equals(links.Count(), zero));
25
26             var setter = new Setter<T,T>(constants.Continue, constants.Break, constants.Null);
27             links.Each(new Link<T>(constants.Any, constants.Any, constants.Any),
        ↪ setter.SetFirstFromListAndReturnTrue);
28
29             Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
30
31             var linkAddress = links.Create();
32
33             var link = new Link<T>(links.GetLink(linkAddress));
34
35             Assert.True(link.Count == 3);
36             Assert.True(equalityComparer.Equals(link.Index, linkAddress));
37             Assert.True(equalityComparer.Equals(link.Source, constants.Null));
38             Assert.True(equalityComparer.Equals(link.Target, constants.Null));
39
40             Assert.True(equalityComparer.Equals(links.Count(), one));
41
42             // Get first link
43             setter = new Setter<T,T>(constants.Continue, constants.Break, constants.Null);
44             links.Each(new Link<T>(constants.Any, constants.Any, constants.Any),
        ↪ setter.SetFirstFromListAndReturnTrue);
45
46             Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
47
48             // Update link to reference itself
49             links.Update(linkAddress, linkAddress, linkAddress);

```

```

50
51     link = new Link<T>(links.GetLink(linkAddress));
52
53     Assert.True(equalityComparer.Equals(link.Source, linkAddress));
54     Assert.True(equalityComparer.Equals(link.Target, linkAddress));
55
56     // Update link to reference null (prepare for delete)
57     var updated = links.Update(linkAddress, constants.Null, constants.Null);
58
59     Assert.True(equalityComparer.Equals(updated, linkAddress));
60
61     link = new Link<T>(links.GetLink(linkAddress));
62
63     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
64     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
65
66     // Delete link
67     links.Delete(linkAddress);
68
69     Assert.True(equalityComparer.Equals(links.Count(), zero));
70
71     setter = new Setter<T,T>(constants.Continue, constants.Break, constants.Null);
72     links.Each(new Link<T>(constants.Any, constants.Any, constants.Any),
73         ↪ setter.SetFirstFromListAndReturnTrue);
74
75     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
76 }
77
78 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
79 {
80     // Constants
81     var constants = links.Constants;
82     var equalityComparer = EqualityComparer<T>.Default;
83
84     var zero = default(T);
85     var one = Arithmetic.Increment(zero);
86     var two = Arithmetic.Increment(one);
87
88     var h106E = new Hybrid<T>(106L, isExternal: true);
89     var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
90     var h108E = new Hybrid<T>(-108L);
91
92     Assert.Equal(106L, h106E.AbsoluteValue);
93     Assert.Equal(107L, h107E.AbsoluteValue);
94     Assert.Equal(108L, h108E.AbsoluteValue);
95
96     // Create Link (External -> External)
97     var linkAddress1 = links.Create();
98
99     links.Update(linkAddress1, h106E, h108E);
100
101     var link1 = new Link<T>(links.GetLink(linkAddress1));
102
103     Assert.True(equalityComparer.Equals(link1.Source, h106E));
104     Assert.True(equalityComparer.Equals(link1.Target, h108E));
105
106     // Create Link (Internal -> External)
107     var linkAddress2 = links.Create();
108
109     links.Update(linkAddress2, linkAddress1, h108E);
110
111     var link2 = new Link<T>(links.GetLink(linkAddress2));
112
113     Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
114     Assert.True(equalityComparer.Equals(link2.Target, h108E));
115
116     // Create Link (Internal -> Internal)
117     var linkAddress3 = links.Create();
118
119     links.Update(linkAddress3, linkAddress1, linkAddress2);
120
121     var link3 = new Link<T>(links.GetLink(linkAddress3));
122
123     Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
124     Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
125
126     // Search for created link
127     var setter1 = new Setter<T, T>(constants.Continue, constants.Break, constants.Null);
128     links.Each(new Link<T>(constants.Any, h106E, h108E),
129         ↪ setter1.SetFirstFromListAndReturnTrue);

```

```

128     Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
129
130     // Search for nonexistent link
131     var setter2 = new Setter<T, T>(constants.Continue, constants.Break, constants.Null);
132     links.Each(new Link<T>(constants.Any, h106E, h108E),
133         ↪ setter1.SetFirstFromListAndReturnTrue);
134
135     Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
136
137     // Update link to reference null (prepare for delete)
138     var updated = links.Update(linkAddress3, constants.Null, constants.Null);
139
140     Assert.True(equalityComparer.Equals(updated, linkAddress3));
141
142     link3 = new Link<T>(links.GetLink(linkAddress3));
143
144     Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
145     Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
146
147     // Delete link
148     links.Delete(linkAddress3);
149
150     Assert.True(equalityComparer.Equals(links.Count(), two));
151
152     var setter3 = new Setter<T, T>(constants.Continue, constants.Break, constants.Null);
153     links.Each(new Link<T>(constants.Any, constants.Any, constants.Any),
154         ↪ setter3.SetFirstFromListAndReturnTrue);
155
156     Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
157 }
158
159 public static void TestMultipleCreationsAndDeletions<TLinkAddress>(this
160     ↪ ILinks<TLinkAddress> links, int numberOfOperations)
161 {
162     for (int i = 0; i < numberOfOperations; i++)
163     {
164         links.Create();
165     }
166     for (int i = 0; i < numberOfOperations; i++)
167     {
168         links.Delete(links.Count());
169     }
170 }
171
172 public static void TestMultipleRandomCreationsAndDeletions<TLinkAddress>(this
173     ↪ ILinks<TLinkAddress> links, int maximumOperationsPerCycle)
174 {
175     var comparer = Comparer<TLinkAddress>.Default;
176     var addressToUInt64Converter = CheckedConverter<TLinkAddress, ulong>.Default;
177     var uint64ToAddressConverter = CheckedConverter<ulong, TLinkAddress>.Default;
178     for (var N = 1; N < maximumOperationsPerCycle; N++)
179     {
180         var random = new System.Random(N);
181         var created = 0UL;
182         var deleted = 0UL;
183         for (var i = 0; i < N; i++)
184         {
185             var linksCount = addressToUInt64Converter.Convert(links.Count());
186             var createPoint = random.NextBoolean();
187             if (linksCount >= 2 && createPoint)
188             {
189                 var linksAddressRange = new Range<ulong>(1, linksCount);
190                 TLinkAddress source = uint64ToAddressConverter.Convert(random.NextUInt64_
191                     ↪ (linksAddressRange));
192                 TLinkAddress target = uint64ToAddressConverter.Convert(random.NextUInt64_
193                     ↪ (linksAddressRange));
194                 ↪ //-V3086
195                 var resultLink = links.GetOrCreate(source, target);
196                 if (comparer.Compare(resultLink,
197                     ↪ uint64ToAddressConverter.Convert(linksCount)) > 0)
198                 {
199                     created++;
200                 }
201             }
202             else
203             {
204                 links.Create();
205                 created++;
206             }
207         }
208     }
209 }

```

```

199     }
200 }
201 Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
202 for (var i = 0; i < N; i++)
203 {
204     TLinkAddress link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
205     if (links.Exists(link))
206     {
207         links.Delete(link);
208         deleted++;
209     }
210 }
211 Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
212 }
213 }
214 }
215 }

```

1.129 ./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         {
19             var linksConstants = new
20                 ↪ LinksConstants<TLinkAddress>(enableExternalReferencesSupport: true);
21             return new UnitedMemoryLinks<TLinkAddress>(new
22                 ↪ FileMappedResizableDirectMemory(dataDBFilename),
23                 ↪ UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
24                 ↪ IndexTreeType.Default);
25         }
26
27         [Fact]
28         public void FormatStructureWithExternalReferenceTest()
29         {
30             ILinks<TLinkAddress> links = CreateLinks();
31             TLinkAddress zero = default;
32             var one = Arithmetic.Increment(zero);
33             var markerIndex = one;
34             var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
35             var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
36                 ↪ markerIndex));
37             AddressToRawNumberConverter<TLinkAddress> addressToNumberConverter = new();
38             var numberAddress = addressToNumberConverter.Convert(1);
39             var numberLink = links.GetOrCreate(numberMarker, numberAddress);
40             var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
41                 ↪ true);
42             Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
43         }
44     }
45 }

```

1.130 ./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().TestMultip
        ↳ leRandomCreationsAndDeletions(100));
29     }
30     private static void Using(Action<ILinks<TLinkAddress>> action)
31     {
32         using (var scope = new Scope<Types<HeapResizableDirectMemory,
        ↳ UInt32UnitedMemoryLinks>>())
33         {
34             action(scope.Use<ILinks<TLinkAddress>>());
35         }
36     }
37 }
38 }

```

1.131 ./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`, 460
`./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`, 464
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs`, 465
`./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs`, 466
`./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs`, 469
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs`, 469
`./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/NoExceptionsDecorator.cs`, 15
`./csharp/Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.cs`, 16
`./csharp/Platform.Data.Doublets/Decorators/UInt32Links.cs`, 17
`./csharp/Platform.Data.Doublets/Decorators/UInt64Links.cs`, 19
`./csharp/Platform.Data.Doublets/Decorators/UniLinks.cs`, 21
`./csharp/Platform.Data.Doublets/Doublet.cs`, 27
`./csharp/Platform.Data.Doublets/DoubletComparer.cs`, 29
`./csharp/Platform.Data.Doublets/FFI/UInt32UnitedMemoryLinks.cs`, 29
`./csharp/Platform.Data.Doublets/FFI/UnitedMemoryLinks.cs`, 31
`./csharp/Platform.Data.Doublets/ILinks.cs`, 40
`./csharp/Platform.Data.Doublets/ILinksExtensions.cs`, 41
`./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs`, 60
`./csharp/Platform.Data.Doublets/Link.cs`, 60
`./csharp/Platform.Data.Doublets/LinkExtensions.cs`, 68
`./csharp/Platform.Data.Doublets/LinksOperatorBase.cs`, 69
`./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs`, 69
`./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs`, 70
`./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs`, 71
`./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`, 91
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 95
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 99
`./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`, 114
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 119
`./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`, 130
`./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

[illegible]

- ./csharp/Platform.Data.Doublets/Stacks/Stack.cs, 442
- ./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs, 443
- ./csharp/Platform.Data.Doublets/SynchronizedLinks.cs, 444
- ./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs, 447
- ./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 450