

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>
25     }

```

```

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(IList<TLinkAddress>? restriction,
52             ↳ IList<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 }
60
61 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the non null contents link deletion resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLinkAddress}"/>
17     public class NonNullContentsLinkDeletionResolver<TLinkAddress> :
18     ↪ LinksDecoratorBase<TLinkAddress>
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="NonNullContentsLinkDeletionResolver"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <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 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5  using TLinkAddress = System.UInt32;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Decorators
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the int 32 links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksDisposableDecoratorBase{TLinkAddress}"/>
18     public class UInt32Links : LinksDisposableDecoratorBase<TLinkAddress>
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt32Links"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public UInt32Links(ILinks<TLinkAddress> links) : base(links) { }
32
33         /// <summary>
34         /// <para>
35         /// Creates the substitution.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="substitution">
40         /// <para>The substitution.</para>
41         /// <para></para>
42         /// </param>
43         /// <returns>
44         /// <para>The link</para>
45         /// <para></para>
46         /// </returns>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

48 public override TLinkAddress Create(IList<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(IList<TLinkAddress>? restriction,
71     ↳ IList<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(IList<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);
125     handlerState.Apply(_links.EnforceResetValues(linkIndex, handlerState.Handler));

```

```

120         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
121         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
122         return handlerState.Result;
123     }
124 }
125 }

```

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

```

1  using System.Collections.Generic;
2  using System.Net.Security;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5  using TLinkAddress = System.UInt64;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Decorators
10 {
11     /// <summary>
12     /// <para>Represents a combined decorator that implements the basic logic for interacting
13     ↪ with the links storage for links with addresses represented as <see cref="System.UInt64"
14     ↪ >.</para>
15     /// <para>Представляет комбинированный декоратор, реализующий основную логику по
16     ↪ взаимодействию с хранилищем связей, для связей с адресами представленными в виде <see
17     ↪ cref="System.UInt64"/>.</para>
18     /// </summary>
19     /// <remarks>
20     /// Возможные оптимизации:
21     /// Объединение в одном поле Source и Target с уменьшением до 32 бит.
22     ///     + меньше объём БД
23     ///     - меньше производительность
24     ///     - больше ограничение на количество связей в БД)
25     /// Ленивое хранение размеров поддеревьев (расчитываемое по мере использования БД)
26     ///     + меньше объём БД
27     ///     - больше сложность
28     ///
29     /// Текущее теоретическое ограничение на индекс связи, из-за использования 5 бит в размере
30     ↪ поддеревьев для AVL баланса и флагов нитей: 2 в степени(64 минус 5 равно 59 ) равно 576
31     ↪ 460 752 303 423 488
32     /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами
33     ↪ (битовыми строками) - вариант матрицы (выстраиваемой лениво).
34     ///
35     /// Решить отключать ли проверки при компиляции под Release. Т.е. исключения будут
36     ↪ выбрасываться только при #if DEBUG
37     /// </remarks>
38     public class UInt64Links : LinksDisposableDecoratorBase<TLinkAddress>
39     {
40         /// <summary>
41         /// <para>
42         /// Initializes a new <see cref="UInt64Links"/> instance.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="links">
47         /// <para>A links.</para>
48         /// <para></para>
49         /// </param>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public UInt64Links(ILinks<TLinkAddress> links) : base(links) { }
52
53         /// <summary>
54         /// <para>
55         /// Creates the substitution.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         /// <param name="substitution">
60         /// <para>The substitution.</para>
61         /// <para></para>
62         /// </param>
63         /// <returns>
64         /// <para>The TLinkAddress</para>
65         /// <para></para>
66         /// </returns>
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         public override TLinkAddress Create(IList<TLinkAddress>? substitution,
69     ↪ WriteHandler<TLinkAddress>? handler) => _links.CreatePoint(handler);

```

```

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(IList<TLinkAddress>? restriction,
82     ↪     IList<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(IList<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));
139         return handlerState.Result;

```

```

135     }
136 }
137 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Platform.Collections;
5  using Platform.Collections.Lists;
6  using Platform.Data.Universal;
7  using Platform.Delegates;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Decorators
12 {
13     /// <remarks>
14     /// What does empty pattern (for condition or substitution) mean? Nothing or Everything?
15     /// Now we go with nothing. And nothing is something one, but empty, and cannot be changed
16     /// by itself. But can cause creation (update from nothing) or deletion (update to nothing).
17     ///
18     /// TODO: Decide to change to IDoubletLinks or not to change. (Better to create
19     ///     DefaultUniLinksBase, that contains logic itself and can be implemented using both
20     ///     IDoubletLinks and ILinks.)
21     /// </remarks>
22     internal class UniLinks<TLinkAddress> : LinksDecoratorBase<TLinkAddress>,
23         IUniLinks<TLinkAddress>
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         /// </param>
37         public UniLinks(ILinks<TLinkAddress> links) : base(links) { }
38         private struct Transition
39         {
40             /// <summary>
41             /// <para>
42             /// The before.
43             /// </para>
44             /// <para></para>
45             /// </summary>
46             public IList<TLinkAddress>? Before;
47             /// <summary>
48             /// <para>
49             /// The after.
50             /// </para>
51             /// <para></para>
52             /// </summary>
53             public IList<TLinkAddress>? After;
54
55             /// <summary>
56             /// <para>
57             /// Initializes a new <see cref="Transition"/> instance.
58             /// </para>
59             /// <para></para>
60             /// </summary>
61             /// <param name="before">
62             /// <para>A before.</para>
63             /// </param>
64             /// <param name="after">
65             /// <para>A after.</para>
66             /// </param>
67             public Transition(IList<TLinkAddress>? before, IList<TLinkAddress>? after)
68             {
69                 Before = before;
70                 After = after;
71             }
72         }
73     }
74 }

```

```
}
```

```
//public static readonly TLinkAddress NullConstant =
```

```
→ Use<LinksConstants<TLinkAddress>>.Single.Null;
```

```
//public static readonly IReadOnlyList<TLinkAddress> NullLink = new
```

```
→ ReadOnlyCollection<TLinkAddress>(new List<TLinkAddress> { NullConstant,
```

```
→ NullConstant, NullConstant });
```

```
// TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
```

```
→ (Links-Expression)
```

```
/// <summary>
```

```
/// <para>
```

```
/// Triggers the restriction.
```

```
/// </para>
```

```
/// <para></para>
```

```
/// </summary>
```

```
/// <param name="restriction">
```

```
/// <para>The restriction.</para>
```

```
/// <para></para>
```

```
/// </param>
```

```
/// <param name="matchedHandler">
```

```
/// <para>The matched handler.</para>
```

```
/// <para></para>
```

```
/// </param>
```

```
/// <param name="substitution">
```

```
/// <para>The substitution.</para>
```

```
/// <para></para>
```

```
/// </param>
```

```
/// <param name="substitutedHandler">
```

```
/// <para>The substituted handler.</para>
```

```
/// <para></para>
```

```
/// </param>
```

```
/// <returns>
```

```
/// <para>The link</para>
```

```
/// <para></para>
```

```
/// </returns>
```

```
public TLinkAddress Trigger(IList<TLinkAddress>? restriction,
```

```
→ WriteHandler<TLinkAddress>? matchedHandler, IList<TLinkAddress>? substitution,
```

```
→ WriteHandler<TLinkAddress>? substitutedHandler)
```

```
{
```

```
    ///List<Transition> transitions = null;
```

```
    ///if (!restriction.IsNullOrEmpty())
```

```
    ///{
```

```
    ///    // Есть причина делать проход (чтение)
```

```
    ///    if (matchedHandler != null)
```

```
    ///    {
```

```
    ///        if (!substitution.IsNullOrEmpty())
```

```
    ///        {
```

```
    ///            // restriction => { 0, 0, 0 } | { 0 } // Create
```

```
    ///            // substitution => { itself, 0, 0 } | { itself, itself, itself } //
```

```
    ///            → Create / Update
```

```
    ///            // substitution => { 0, 0, 0 } | { 0 } // Delete
```

```
    ///            transitions = new List<Transition>();
```

```
    ///            if (Equals(substitution[Constants.IndexPart], Constants.Null))
```

```
    ///            {
```

```
    ///                // If index is Null, that means we always ignore every other
```

```
    ///                → value (they are also Null by definition)
```

```
    ///                var matchDecision = matchedHandler(, NullLink);
```

```
    ///                if (Equals(matchDecision, Constants.Break))
```

```
    ///                return false;
```

```
    ///                if (!Equals(matchDecision, Constants.Skip))
```

```
    ///                transitions.Add(new Transition(matchedLink, newValue));
```

```
    ///            }
```

```
    ///            else
```

```
    ///            {
```

```
    ///                Func<T, bool> handler;
```

```
    ///                handler = link =>
```

```
    ///                {
```

```
    ///                    var matchedLink = Memory.GetLinkValue(link);
```

```
    ///                    var newValue = Memory.GetLinkValue(link);
```

```
    ///                    newValue[Constants.IndexPart] = Constants.Itself;
```

```
    ///                    newValue[Constants.SourcePart] =
```

```
    ///                    → Equals(substitution[Constants.SourcePart], Constants.Itself) ?
```

```
    ///                    → matchedLink[Constants.IndexPart] : substitution[Constants.SourcePart];
```

```
    ///                    newValue[Constants.TargetPart] =
```

```
    ///                    → Equals(substitution[Constants.TargetPart], Constants.Itself) ?
```

```
    ///                    → matchedLink[Constants.IndexPart] : substitution[Constants.TargetPart];
```

```
    ///                }
```

```
    ///            }
```

```

135         var matchDecision = matchedHandler(matchedLink, newValue);
136         if (Equals(matchDecision, Constants.Break))
137             return false;
138         if (!Equals(matchDecision, Constants.Skip))
139             transitions.Add(new Transition(matchedLink, newValue));
140         return true;
141     };
142     if (!Memory.Each(handler, restriction))
143         return Constants.Break;
144     }
145     }
146     else
147     {
148         Func<T, bool> handler = link =>
149         {
150             var matchedLink = Memory.GetLinkValue(link);
151             var matchDecision = matchedHandler(matchedLink, matchedLink);
152             return !Equals(matchDecision, Constants.Break);
153         };
154         if (!Memory.Each(handler, restriction))
155             return Constants.Break;
156     }
157 }
158 else
159 {
160     if (substitution != null)
161     {
162         transitions = new List<IList<T>>();
163         Func<T, bool> handler = link =>
164         {
165             var matchedLink = Memory.GetLinkValue(link);
166             transitions.Add(matchedLink);
167             return true;
168         };
169         if (!Memory.Each(handler, restriction))
170             return Constants.Break;
171     }
172     else
173     {
174         return Constants.Continue;
175     }
176 }
177 }
178 if (substitution != null)
179 {
180     // Есть причина делать замену (запись)
181     if (substitutedHandler != null)
182     {
183     }
184     else
185     {
186     }
187 }
188 return Constants.Continue;
189
190 //if (restriction.IsNullOrEmpty()) // Create
191 //{
192 //    substitution[Constants.IndexPart] = Memory.AllocateLink();
193 //    Memory.SetLinkValue(substitution);
194 //}
195 //else if (substitution.IsNullOrEmpty()) // Delete
196 //{
197 //    Memory.FreeLink(restriction[Constants.IndexPart]);
198 //}
199 //else if (restriction.EqualTo(substitution)) // Read or ("repeat" the state) // Each
200 //{
201 //    // No need to collect links to list
202 //    // Skip == Continue
203 //    // No need to check substitutedHandler
204 //    if (!Memory.Each(link => !Equals(matchedHandler(Memory.GetLinkValue(link)),
205 //        ↪ Constants.Break), restriction))
206 //        return Constants.Break;
207 //}
208 //else // Update
209 //{
210 //    //List<IList<T>> matchedLinks = null;
211 //    if (matchedHandler != null)
212 //    {

```

```

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>
286 /// <exception cref="NotSupportedException">
287 /// <para></para>
288 /// </exception>

```

```

287     /// <para></para>
288     /// </exception>
289     /// <exception cref="NotSupportedException">
290     /// <para></para>
291     /// <para></para>
292     /// </exception>
293     /// <exception cref="NotSupportedException">
294     /// <para></para>
295     /// <para></para>
296     /// </exception>
297     /// <returns>
298     /// <para>The link</para>
299     /// <para></para>
300     /// </returns>
301 public TLinkAddress Trigger(IList<TLinkAddress>? patternOrCondition,
    ↳ ReadHandler<TLinkAddress>? matchHandler, IList<TLinkAddress>? substitution,
    ↳ WriteHandler<TLinkAddress>? substitutionHandler)
302 {
303     var constants = _constants;
304     if (patternOrCondition.IsNullOrEmpty() && substitution.IsNullOrEmpty())
305     {
306         return constants.Continue;
307     }
308     else if (patternOrCondition.EqualTo(substitution)) // Should be Each here TODO:
    ↳ Check if it is a correct condition
309     {
310         // Or it only applies to trigger without matchHandler.
311         throw new NotImplementedException();
312     }
313     else if (!substitution.IsNullOrEmpty()) // Creation
314     {
315         var before = Array.Empty<TLinkAddress>();
316         // Что должно означать False здесь? Остановиться (перестать идти) или пропустить
    ↳ (пройти мимо) или пустить (взять)?
317         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
    ↳ constants.Break))
318         {
319             return constants.Break;
320         }
321         var after = (IList<TLinkAddress>?)substitution.ToArray();
322         if (_equalityComparer.Equals(after[0], default))
323         {
324             var newLink = _links.Create();
325             after[0] = newLink;
326         }
327         if (substitution.Count == 1)
328         {
329             after = _links.GetLink(substitution[0]);
330         }
331         else if (substitution.Count == 3)
332         {
333             //Links.Create(after);
334         }
335         else
336         {
337             throw new NotSupportedException();
338         }
339         return matchHandler != null ? substitutionHandler(before, after) :
    ↳ constants.Continue;
340     }
341     else if (!patternOrCondition.IsNullOrEmpty()) // Deletion
342     {
343         if (patternOrCondition.Count == 1)
344         {
345             var linkToDelete = patternOrCondition[0];
346             var before = _links.GetLink(linkToDelete);
347             if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
    ↳ constants.Break))
348             {
349                 return constants.Break;
350             }
351             var after = Array.Empty<TLinkAddress>();
352             _links.Update(linkToDelete, constants.Null, constants.Null);
353             _links.Delete(linkToDelete);
354             return matchHandler != null ? substitutionHandler(before, after) :
    ↳ constants.Continue;
355         }
356         else

```



```

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>
79         /// <para>.</para>
80         /// </summary>
81         /// <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(Doulet<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 Doulet<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 ==(Doulet<T> left, Doulet<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>
156   /// <para>.</para>
157   /// </param>
158   /// <returns>

```

```

157     /// <para>.</para>
158     /// <para>.</para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public static bool operator !=(Doublet<T> left, Doublet<T> right) => !(left == right);
162 }
163 }

```

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

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets
7 {
8     /// <remarks>
9     /// TODO: Может стоит попробовать ref во всех методах (IRefEqualityComparer)
10    /// 2x faster with comparer
11    /// </remarks>
12    public class DoubletComparer<T> : IEqualityComparer<Doublet<T>>
13    {
14        /// <summary>
15        /// <para>
16        /// The .
17        /// </para>
18        /// <para></para>
19        /// </summary>
20        public static readonly DoubletComparer<T> Default = new DoubletComparer<T>();
21
22        /// <summary>
23        /// <para>
24        /// Determines whether this instance equals.
25        /// </para>
26        /// <para></para>
27        /// </summary>
28        /// <param name="x">
29        /// <para>The .</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="y">
33        /// <para>The .</para>
34        /// <para></para>
35        /// </param>
36        /// <returns>
37        /// <para>The bool</para>
38        /// <para></para>
39        /// </returns>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]
41        public bool Equals(Doublet<T> x, Doublet<T> y) => x.Equals(y);
42
43        /// <summary>
44        /// <para>
45        /// Gets the hash code using the specified obj.
46        /// </para>
47        /// <para></para>
48        /// </summary>
49        /// <param name="obj">
50        /// <para>The obj.</para>
51        /// <para></para>
52        /// </param>
53        /// <returns>
54        /// <para>The int</para>
55        /// <para></para>
56        /// </returns>
57        [MethodImpl(MethodImplOptions.AggressiveInlining)]
58        public int GetHashCode(Doublet<T> obj) => obj.GetHashCode();
59    }
60 }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.InteropServices;
5 using Platform.Converters;
6 using Platform.Delegates;
7 using Platform.Disposables;
8

```

```

9 namespace Platform.Data.Doublets.FFI
10 {
11     using TLinkAddress = System.UInt32;
12
13     public class UInt32UnitedMemoryLinks : DisposableBase, ILinks<TLinkAddress>
14     {
15         public LinksConstants<TLinkAddress> Constants { get; }
16
17         private readonly unsafe void* _ptr;
18
19         public UInt32UnitedMemoryLinks(string path)
20         {
21             unsafe
22             {
23                 _ptr = Methods.UInt32UnitedMemoryLinks_New(path);
24
25                 // TODO: Update api
26                 Constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
27                     ↪ true);
28             }
29
30             public TLinkAddress Count(IList<TLinkAddress>? restriction)
31             {
32                 unsafe
33                 {
34                     var array = stackalloc uint[restriction.Count];
35                     for (var i = 0; i < restriction.Count; i++)
36                     {
37                         array[i] = restriction[i];
38                     }
39                     return Methods.UInt32UnitedMemoryLinks_Count(_ptr, array,
40                         ↪ (nuint)(restriction?.Count ?? 0));
41                 }
42
43                 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
82                     {
83                         var restrictionArray = stackalloc uint[restriction.Count];
84                         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 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.InteropServices;
5  using Platform.Converters;
6  using Platform.Delegates;
7  using Platform.Disposables;
8
9  namespace Platform.Data.Doublets.FFI
10 {
11     struct FfiLink_UInt8
12     {
13         public Byte Index;
14         public Byte Source;
15         public Byte Target;
16     }
17
18     struct FfiLink_UInt16
19     {
20         public UInt16 Index;
21         public UInt16 Source;
22         public UInt16 Target;
23     }
24
25     struct FfiLink_UInt32
26     {
27         public UInt32 Index;

```

```

28     public UInt32 Source;
29     public UInt32 Target;
30 }
31
32 struct FfiLink_UInt64
33 {
34     public UInt64 Index;
35     public UInt64 Source;
36     public UInt64 Target;
37 }
38
39 unsafe static class Methods
40 {
41     private const string DllName = "Platform.Doublets";
42
43     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
44     public delegate Byte EachCallback_UInt8(FfiLink_UInt8 link);
45
46     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
47     public delegate UInt16 EachCallback_UInt16(FfiLink_UInt16 link);
48
49     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
50     public delegate UInt32 EachCallback_UInt32(FfiLink_UInt32 link);
51
52     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
53     public delegate UInt64 EachCallback_UInt64(FfiLink_UInt64 link);
54
55     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
56     public delegate Byte CreateCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
57
58     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
59     public delegate UInt16 CreateCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
60         ↪ after);
61
62     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
63     public delegate UInt32 CreateCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
64         ↪ after);
65
66     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
67     public delegate UInt64 CreateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
68         ↪ after);
69
70     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
71     public delegate Byte UpdateCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
72
73     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
74     public delegate UInt16 UpdateCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
75         ↪ after);
76
77     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
78     public delegate UInt32 UpdateCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
79         ↪ after);
80
81     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
82     public delegate UInt64 UpdateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
83         ↪ after);
84
85     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
86     public delegate Byte DeleteCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
87
88     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
89     public delegate UInt16 DeleteCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
90         ↪ after);
91
92     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
93     public delegate UInt32 DeleteCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
94         ↪ after);
95
96     [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
97     public delegate UInt64 DeleteCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
98         ↪ after);
99
100     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
101     public static extern void* ByteUnitedMemoryLinks_New(string path);
102
103     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
104     public static extern void* UInt16UnitedMemoryLinks_New(string path);
105
106     [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);
172
173 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]

```



```

158     public static extern uint UInt32UnitedMemoryLinks_Update(void* self, uint* restriction,
159         ↳ nuint restrictionLength, uint* substitution, nuint substitutionLength,
160         ↳ UpdateCallback UInt32 callback);
161
162     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
163     public static extern ulong UInt64UnitedMemoryLinks_Update(void* self, ulong*
164         ↳ restriction, nuint restrictionLength, ulong* substitution, nuint
165         ↳ substitutionLength, UpdateCallback UInt64 callback);
166
167     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
168     public static extern byte ByteUnitedMemoryLinks_Delete(void* self, byte* restriction,
169         ↳ nuint restrictionLength, DeleteCallback UInt8 callback);
170
171     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
172     public static extern ushort UInt16UnitedMemoryLinks_Delete(void* self, ushort*
173         ↳ restriction, nuint len, DeleteCallback UInt16 callback);
174
175     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
176     public static extern uint UInt32UnitedMemoryLinks_Delete(void* self, uint* restriction,
177         ↳ nuint len, DeleteCallback UInt32 callback);
178
179     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
180     public static extern ulong UInt64UnitedMemoryLinks_Delete(void* self, ulong*
181         ↳ restriction, nuint len, DeleteCallback UInt64 callback);
182 }
183
184 public class UnitedMemoryLinks<TLinkAddress> : DisposableBase, ILinks<TLinkAddress>
185 {
186     private static readonly UncheckedConverter<byte, TLinkAddress> from_u8 =
187         ↳ UncheckedConverter<byte, TLinkAddress>.Default;
188     private static readonly UncheckedConverter<ushort, TLinkAddress> from_u16 =
189         ↳ UncheckedConverter<ushort, TLinkAddress>.Default;
190     private static readonly UncheckedConverter<uint, TLinkAddress> from_u32 =
191         ↳ UncheckedConverter<uint, TLinkAddress>.Default;
192     private static readonly UncheckedConverter<ulong, TLinkAddress> from_u64 =
193         ↳ UncheckedConverter<ulong, TLinkAddress>.Default;
194     private static readonly UncheckedConverter<TLinkAddress, ulong> from_t =
195         ↳ UncheckedConverter<TLinkAddress, ulong>.Default;
196
197     public LinksConstants<TLinkAddress> Constants { get; }
198
199     private readonly unsafe void* _ptr;
200
201     public UnitedMemoryLinks(string path)
202     {
203         TLinkAddress t = default;
204         unsafe
205         {
206             _ptr = t switch
207             {
208                 byte => Methods.ByteUnitedMemoryLinks_New(path),
209                 ushort => Methods.UInt16UnitedMemoryLinks_New(path),
210                 uint => Methods.UInt32UnitedMemoryLinks_New(path),
211                 ulong => Methods.UInt64UnitedMemoryLinks_New(path),
212                 _ => throw new NotImplementedException()
213             };
214
215             // TODO: Update api
216             Constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
217                 ↳ true);
218         }
219     }
220
221     public TLinkAddress Count(ICollection<TLinkAddress>? restriction)
222     {
223         var restrictionLength = restriction?.Count ?? 0;
224         unsafe
225         {
226             TLinkAddress t = default;
227             switch (t)
228             {
229                 case byte:
230                 {
231                     var restrictionArray = stackalloc byte[restrictionLength];
232                     var byteRestrictionArray = (ICollection<byte>)restriction;
233                     for (var i = 0; i < restrictionLength; i++)
234                     {
235                         restrictionArray[i] = byteRestrictionArray[i];
236                     }
237                 }
238             }
239         }
240     }
241 }

```

```

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 public TLinkAddress Each(IList<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
272 ↪ handler)
273 {
274     var restrictionLength = restriction?.Count ?? 0;
275     unsafe
276     {
277         TLinkAddress t = default;
278         switch (t)
279         {
280             case byte:
281             {
282                 byte Callback(FfiLink_UInt8 link) => (byte)from_t.Convert(handler !=
283                               ↪ null ? handler(new Link<TLinkAddress>(from_u8.Convert(link.Index),
284                               ↪ from_u8.Convert(link.Source), from_u8.Convert(link.Target))) :
285                               ↪ Constants.Continue);
286                 var restrictionArray = stackalloc byte[restrictionLength];
287                 var byteRestrictionArray = (IList<byte>)restriction;
288                 for (var i = 0; i < restrictionLength; i++)
289                 {
290                     restrictionArray[i] = byteRestrictionArray[i];
291                 }
292                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Each(_ptr,
293                               ↪ restrictionArray, (nuint)restrictionLength, Callback));
294             }
295             case ushort:
296             {
297                 ushort Callback(FfiLink_UInt16 link) => (ushort)from_t.Convert(handler
298                               ↪ != null ? handler(new
299                               ↪ Link<TLinkAddress>(from_u16.Convert(link.Index),
300                               ↪ from_u16.Convert(link.Source), from_u16.Convert(link.Target))) :
301                               ↪ Constants.Continue);
302                 var restrictionArray = stackalloc ushort[restrictionLength];

```

```

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

```

```

348     ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
349         ↳ (ushort)from_t.Convert(handler != null ? handler(new
350         ↳ Link<TLinkAddress>(from_u16.Convert(before.Index),
351         ↳ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
352         ↳ new Link<TLinkAddress>(from_u16.Convert(after.Index),
353         ↳ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
354         ↳ Constants.Continue);
355     var substitutionArray = stackalloc ushort[substitutionLength];
356     var ushortSubstitutionArray = (IList<ushort>)substitution;
357     for (var i = 0; i < substitutionLength; i++)
358     {
359         substitutionArray[i] = ushortSubstitutionArray[i];
360     }
361     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Create(_ptr,
362     ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
363 }
364 case uint:
365 {
366     uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
367         ↳ (uint)from_t.Convert(handler != null ? handler(new
368         ↳ Link<TLinkAddress>(from_u32.Convert(before.Index),
369         ↳ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
370         ↳ new Link<TLinkAddress>(from_u32.Convert(after.Index),
371         ↳ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) :
372         ↳ Constants.Continue);
373     var substitutionArray = stackalloc uint[substitutionLength];
374     var uintSubstitutionArray = (IList<uint>)substitution;
375     for (var i = 0; i < substitutionLength; i++)
376     {
377         substitutionArray[i] = uintSubstitutionArray[i];
378     }
379     return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Create(_ptr,
380     ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
381 }
382 case ulong:
383 {
384     ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
385         ↳ (ulong)from_t.Convert(handler != null ? handler(new
386         ↳ Link<TLinkAddress>(from_u64.Convert(before.Index),
387         ↳ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
388         ↳ new Link<TLinkAddress>(from_u64.Convert(after.Index),
389         ↳ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
390         ↳ Constants.Continue);
391     var substitutionArray = stackalloc ulong[substitutionLength];
392     var ulongSubstitutionArray = (IList<ulong>)substitution;
393     for (var i = 0; i < substitutionLength; i++)
394     {
395         substitutionArray[i] = ulongSubstitutionArray[i];
396     }
397     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Create(_ptr,
398     ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
399 }
400 default:
401 {
402     throw new NotImplementedException();
403 }
404 };
405 }
406 }
407 }
408
409 public TLinkAddress Update(IList<TLinkAddress>? restriction, IList<TLinkAddress>?
410 ↳ substitution, WriteHandler<TLinkAddress>? handler)
411 {
412     var restrictionLength = restriction?.Count ?? 0;
413     var substitutionLength = substitution?.Count ?? 0;
414     unsafe
415     {
416         TLinkAddress t = default;
417         switch (t)
418         {
419             case byte:
420             {
421                 var restrictionArray = stackalloc byte[restrictionLength];
422                 var byteRestrictionArray = (IList<byte>)restriction;
423                 for (var i = 0; i < restrictionLength; i++)
424                 {

```

```

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     {
414         (byte)from_t.Convert(handler != null ? handler(new
415             ↳ Link<TLinkAddress>(from_u8.Convert(before.Index),
416             ↳ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
417             ↳ Link<TLinkAddress>(from_u8.Convert(after.Index),
418             ↳ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
419             ↳ Constants.Continue);
420         return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Update(_ptr,
421             ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
422             ↳ (nuint)(substitution?.Count ?? 0), Callback));
423     }
424     case ushort:
425     {
426         var restrictionArray = stackalloc ushort[restrictionLength];
427         var ushortRestrictionArray = (IList<ushort>)restriction;
428         for (var i = 0; i < restrictionLength; i++)
429         {
430             restrictionArray[i] = ushortRestrictionArray[i];
431         }
432         var substitutionArray = stackalloc ushort[substitutionLength];
433         var ushortSubstitutionArray = (IList<ushort>)substitution;
434         for (var i = 0; i < substitutionLength; i++)
435         {
436             substitutionArray[i] = ushortSubstitutionArray[i];
437         }
438         ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
439         {
440             (ushort)from_t.Convert(handler != null ? handler(new
441                 ↳ Link<TLinkAddress>(from_u16.Convert(before.Index),
442                 ↳ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
443                 ↳ new Link<TLinkAddress>(from_u16.Convert(after.Index),
444                 ↳ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) :
445                 ↳ Constants.Continue);
446             return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Update(_ptr,
447                 ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
448                 ↳ (nuint)(substitution?.Count ?? 0), Callback));
449         }
450     case uint:
451     {
452         var restrictionArray = stackalloc uint[restrictionLength];
453         var uintRestrictionArray = (IList<uint>)restriction;
454         for (var i = 0; i < restrictionLength; i++)
455         {
456             restrictionArray[i] = uintRestrictionArray[i];
457         }
458         var substitutionArray = stackalloc uint[substitutionLength];
459         var uintSubstitutionArray = (IList<uint>)substitution;
460         for (var i = 0; i < substitutionLength; i++)
461         {
462             substitutionArray[i] = uintSubstitutionArray[i];
463         }
464         uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
465         {
466             (uint)from_t.Convert(handler != null ? handler(new
467                 ↳ Link<TLinkAddress>(from_u32.Convert(before.Index),
468                 ↳ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
469                 ↳ new Link<TLinkAddress>(from_u32.Convert(after.Index),
470                 ↳ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) :
471                 ↳ Constants.Continue);
472             return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Update(_ptr,
473                 ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
474                 ↳ (nuint)(substitution?.Count ?? 0), Callback));
475         }
476     case ulong:
477     {
478         var restrictionArray = stackalloc ulong[restrictionLength];
479         var ulongRestrictionArray = (IList<ulong>)restriction;
480         for (var i = 0; i < restrictionLength; i++)
481         {
482             restrictionArray[i] = ulongRestrictionArray[i];
483         }
484     }

```

```

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     {
465         (ulong)from_t.Convert(handler != null ? handler(new
466             ↪ Link<TLinkAddress>(from_u64.Convert(before.Index),
467             ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
468             ↪ new Link<TLinkAddress>(from_u64.Convert(after.Index),
469             ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) :
470             ↪ Constants.Continue);
471     }
472     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Update(_ptr,
473         ↪ restrictionArray, (nuint)restrictionLength, substitutionArray,
474         ↪ (nuint)(substitution?.Count ?? 0), Callback));
475 }
476 default:
477 {
478     throw new NotImplementedException();
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 }

public TLinkAddress Delete(IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
    ↪ handler)
{
    var restrictionLength = restriction?.Count ?? 0;
    unsafe
    {
        TLinkAddress t = default;
        switch (t)
        {
            case byte:
            {
                var restrictionArray = stackalloc byte[restrictionLength];
                var byteRestrictionArray = (IList<byte>)restriction;
                for (var i = 0; i < restrictionLength; i++)
                {
                    restrictionArray[i] = byteRestrictionArray[i];
                }
                byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
                {
                    (byte)from_t.Convert(handler != null ? handler(new
                    ↪ Link<TLinkAddress>(from_u8.Convert(before.Index),
                    ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
                    ↪ Link<TLinkAddress>(from_u8.Convert(after.Index),
                    ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) :
                    ↪ Constants.Continue);
                }
                return (TLinkAddress)(object)Methods.ByteUnitedMemoryLinks_Delete(_ptr,
                    ↪ restrictionArray, (nuint)restrictionLength, Callback);
            }
            case ushort:
            {
                var restrictionArray = stackalloc ushort[restrictionLength];
                var ushortRestrictionArray = (IList<ushort>)restriction;
                for (var i = 0; i < restrictionLength; i++)
                {
                    restrictionArray[i] = ushortRestrictionArray[i];
                }
                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);
                }
                return
                    ↪ (TLinkAddress)(object)Methods.UInt16UnitedMemoryLinks_Delete(_ptr,
                    ↪ restrictionArray, (nuint)restrictionLength, Callback);
            }
            case uint:
            {
                var restrictionArray = stackalloc uint[restrictionLength];
                var uintRestrictionArray = (IList<uint>)restriction;
                for (var i = 0; i < restrictionLength; i++)
                {
                    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 }

```

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

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  using System.Collections.Generic;
4
5  namespace Platform.Data.Doublets
6  {
7      /// <summary>
8      /// <para>

```

```

9      /// Defines the links.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="ILinks{TLinkAddress, LinksConstants{TLinkAddress}}"/>
14     public interface ILinks<TLinkAddress> : ILinks<TLinkAddress, LinksConstants<TLinkAddress>>
15     {
16     }
17 }

```

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

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Runtime.CompilerServices;
6  using Platform.Ranges;
7  using Platform.Collections.Lists;
8  using Platform.Random;
9  using Platform.Setters;
10 using Platform.Converters;
11 using Platform.Numbers;
12 using Platform.Data.Exceptions;
13 using Platform.Data.Doublets.Decorators;
14 using Platform.Delegates;
15
16 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18 namespace Platform.Data.Doublets
19 {
20     /// <summary>
21     /// <para>
22     /// Represents the links extensions.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     public static class ILinksExtensions
27     {
28         /// <summary>
29         /// <para>
30         /// Runs the random creations using the specified links.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <typeparam name="TLinkAddress">
35         /// <para>The link.</para>
36         /// <para></para>
37         /// </typeparam>
38         /// <param name="links">
39         /// <para>The links.</para>
40         /// <para></para>
41         /// </param>
42         /// <param name="amountOfCreations">
43         /// <para>The amount of creations.</para>
44         /// <para></para>
45         /// </param>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public static void RunRandomCreations<TLinkAddress>(this ILinks<TLinkAddress> links,
48             ↪ ulong amountOfCreations)
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.
68         /// </para>
69     }

```



```

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

```

```

359 /// </param>
360 /// <exception cref="ArgumentOutOfRangeException">
361 /// <para>Sequences with sizes other than powers of two are not supported.</para>
362 /// </exception>
363 /// <returns>
364 /// <para>The current link.</para>
365 /// </returns>
366 [MethodImpl(MethodImplOptions.AggressiveInlining)]
367 public static TLinkAddress GetSquareMatrixSequenceElementByIndex<TLinkAddress>(this
368     ↳ ILinks<TLinkAddress> links, TLinkAddress root, ulong size, ulong index)
369 {
370     var constants = links.Constants;
371     var source = constants.SourcePart;
372     var target = constants.TargetPart;
373     if (!Platform.Numbers.Math.IsPowerOfTwo(size))
374     {
375         throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
376             ↳ than powers of two are not supported.");
377     }
378     var path = new BitArray(BitConverter.GetBytes(index));
379     var length = Bit.GetLowestPosition(size);
380     links.EnsureLinkExists(root, "root");
381     var currentLink = root;
382     for (var i = length - 1; i >= 0; i--)
383     {
384         currentLink = links.GetLink(currentLink)[path[i] ? target : source];
385     }
386     return currentLink;
387 }
388
389 #endregion
390
391 /// <summary>
392 /// Возвращает индекс указанной связи.
393 /// </summary>
394 /// <param name="links">Хранилище связей.</param>
395 /// <param name="link">Связь представленная списком, состоящим из её адреса и
396     ↳ содержимого.</param>
397 /// <returns>Индекс начальной связи для указанной связи.</returns>
398 [MethodImpl(MethodImplOptions.AggressiveInlining)]
399 public static TLinkAddress GetIndex<TLinkAddress>(this ILinks<TLinkAddress> links,
400     ↳ IList<TLinkAddress>? link) => link[links.Constants.IndexPart];
401
402 /// <summary>
403 /// Возвращает индекс начальной (Source) связи для указанной связи.
404 /// </summary>
405 /// <param name="links">Хранилище связей.</param>
406 /// <param name="link">Индекс связи.</param>
407 /// <returns>Индекс начальной связи для указанной связи.</returns>
408 [MethodImpl(MethodImplOptions.AggressiveInlining)]
409 public static TLinkAddress GetSource<TLinkAddress>(this ILinks<TLinkAddress> links,
410     ↳ TLinkAddress link) => links.GetLink(link)[links.Constants.SourcePart];
411
412 /// <summary>
413 /// Возвращает индекс начальной (Source) связи для указанной связи.
414 /// </summary>
415 /// <param name="links">Хранилище связей.</param>
416 /// <param name="link">Связь представленная списком, состоящим из её адреса и
417     ↳ содержимого.</param>
418 /// <returns>Индекс начальной связи для указанной связи.</returns>
419 [MethodImpl(MethodImplOptions.AggressiveInlining)]
420 public static TLinkAddress GetSource<TLinkAddress>(this ILinks<TLinkAddress> links,
421     ↳ IList<TLinkAddress>? link) => link[links.Constants.SourcePart];
422
423 /// <summary>
424 /// Возвращает индекс конечной (Target) связи для указанной связи.
425 /// </summary>
426 /// <param name="links">Хранилище связей.</param>
427 /// <param name="link">Индекс связи.</param>
428 /// <returns>Индекс конечной связи для указанной связи.</returns>
429 [MethodImpl(MethodImplOptions.AggressiveInlining)]
430 public static TLinkAddress GetTarget<TLinkAddress>(this ILinks<TLinkAddress> links,
431     ↳ TLinkAddress link) => links.GetLink(link)[links.Constants.TargetPart];
432
433 /// <summary>
434 /// 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 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
438 /// </summary>
439 /// <param name="links">Хранилище связей.</param>
440 /// <param name="handler">Обработчик каждой подходящей связи.</param>
441 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
    ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Any -
    ↳ отсутствие ограничения, 1..∞ конкретный адрес связи.</param>
442 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
443 [MethodImpl(MethodImplOptions.AggressiveInlining)]
444 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ ReadHandler<TLinkAddress>? handler, params TLinkAddress[] restriction)
    ↳ => links.Each(handler, (IList<TLinkAddress>)restriction);
445
446 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ ReadHandler<TLinkAddress>? handler, IList<TLinkAddress> restriction)
447 => EqualityComparer<TLinkAddress>.Default.Equals(links.Each(restriction, handler),
    ↳ links.Constants.Continue);
448
449 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ Func<TLinkAddress, bool> handler, TLinkAddress source, TLinkAddress target) =>
450 links.Each(source, target, handler);
451
452
453 /// <summary>
454 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
455 /// </summary>
456 /// <param name="links">Хранилище связей.</param>
457 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
    ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
458 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
    ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
459 /// <param name="handler">Обработчик каждой подходящей связи.</param>
460 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
461 [MethodImpl(MethodImplOptions.AggressiveInlining)]
462 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
    ↳ source, TLinkAddress target, Func<TLinkAddress, bool> handler)
463 {
464     var constants = links.Constants;
465     return links.Each(link => handler(links.GetIndex(link)) ? constants.Continue :
    ↳ constants.Break, constants.Any, source, target);
466 }
467
468 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ ReadHandler<TLinkAddress>? handler, TLinkAddress source, TLinkAddress target) =>
469 links.Each(source, target, handler);
470
471
472 /// <summary>
473 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
474 /// </summary>
475 /// <param name="links">Хранилище связей.</param>
476 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
    ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
477 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
    ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
    ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
    ↳ <param name="handler">Обработчик каждой подходящей связи.</param>

```

```

478 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
479   ↳ случае.</returns>
480 [MethodImpl(MethodImplOptions.AggressiveInlining)]
481 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
482   ↳ source, TLinkAddress target, ReadHandler<TLinkAddress>? handler) =>
483   ↳ links.Each(handler, links.Constants.Any, source, target);
484
485 /// <summary>
486 /// <para>
487 /// Alls the links.
488 /// </para>
489 /// <para></para>
490 /// </summary>
491 /// <typeparam name="TLinkAddress">
492 /// <para>The link.</para>
493 /// <para></para>
494 /// </typeparam>
495 /// <param name="links">
496 /// <para>The links.</para>
497 /// <para></para>
498 /// </param>
499 /// <param name="restriction">
500 /// <para>The restriction.</para>
501 /// <para></para>
502 /// </param>
503 /// <returns>
504 /// <para>A list of i list t link</para>
505 /// <para></para>
506 /// </returns>
507 [MethodImpl(MethodImplOptions.AggressiveInlining)]
508 public static IList<IList<TLinkAddress>?> All<TLinkAddress>(this ILinks<TLinkAddress>
509   ↳ links, params TLinkAddress[] restriction)
510 {
511     var allLinks = new List<IList<TLinkAddress>?>();
512     var filler = new ListFiller<IList<TLinkAddress>?, TLinkAddress>(allLinks,
513       ↳ links.Constants.Continue);
514     links.Each(filler.AddAndReturnConstant, restriction);
515     return allLinks;
516 }
517
518 /// <summary>
519 /// <para>
520 /// Alls the indices using the specified links.
521 /// </para>
522 /// <para></para>
523 /// </summary>
524 /// <typeparam name="TLinkAddress">
525 /// <para>The link.</para>
526 /// <para></para>
527 /// </typeparam>
528 /// <param name="links">
529 /// <para>The links.</para>
530 /// <para></para>
531 /// </param>
532 /// <param name="restriction">
533 /// <para>The restriction.</para>
534 /// <para></para>
535 /// </param>
536 /// <returns>
537 /// <para>A list of t link</para>
538 /// <para></para>
539 /// </returns>
540 [MethodImpl(MethodImplOptions.AggressiveInlining)]
541 public static IList<TLinkAddress>? AllIndices<TLinkAddress>(this ILinks<TLinkAddress>
542   ↳ links, params TLinkAddress[] restriction)
543 {
544     var allIndices = new List<TLinkAddress>();
545     var filler = new ListFiller<TLinkAddress, TLinkAddress>(allIndices,
546       ↳ links.Constants.Continue);
547     links.Each(filler.AddFirstAndReturnConstant, restriction);
548     return allIndices;
549 }
550
551 /// <summary>
552 /// Возвращает значение, определяющее существует ли связь с указанными началом и концом
553   ↳ в хранилище связей.
554 /// </summary>

```

```

547 /// <param name="links">Хранилище связей.</param>
548 /// <param name="source">Начало связи.</param>
549 /// <param name="target">Конец связи.</param>
550 /// <returns>Значение, определяющее существует ли связь.</returns>
551 [MethodImpl(MethodImplOptions.AggressiveInlining)]
552 public static bool Exists<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
    ↳ source, TLinkAddress target) =>
    ↳ Comparer<TLinkAddress>.Default.Compare(links.Count(links.Constants.Any, source,
    ↳ target), default) > 0;

553
554 #region Ensure
555 // TODO: May be move to EnsureExtensions or make it both there and here
556
557 /// <summary>
558 /// <para>
559 /// Ensures the link exists using the specified links.
560 /// </para>
561 /// <para></para>
562 /// </summary>
563 /// <typeparam name="TLinkAddress">
564 /// <para>The link.</para>
565 /// <para></para>
566 /// </typeparam>
567 /// <param name="links">
568 /// <para>The links.</para>
569 /// <para></para>
570 /// </param>
571 /// <param name="restriction">
572 /// <para>The restriction.</para>
573 /// <para></para>
574 /// </param>
575 /// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
576 /// <para>sequence[{i}]</para>
577 /// <para></para>
578 /// </exception>
579 [MethodImpl(MethodImplOptions.AggressiveInlining)]
580 public static void EnsureLinkExists<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↳ IList<TLinkAddress>? restriction)
581 {
582     for (var i = 0; i < restriction.Count; i++)
583     {
584         if (!links.Exists(restriction[i]))
585         {
586             throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(restriction[i],
    ↳ $"sequence[{i}]");
587         }
588     }
589 }
590
591 /// <summary>
592 /// <para>
593 /// Ensures the inner reference exists using the specified links.
594 /// </para>
595 /// <para></para>
596 /// </summary>
597 /// <typeparam name="TLinkAddress">
598 /// <para>The link.</para>
599 /// <para></para>
600 /// </typeparam>
601 /// <param name="links">
602 /// <para>The links.</para>
603 /// <para></para>
604 /// </param>
605 /// <param name="reference">
606 /// <para>The reference.</para>
607 /// <para></para>
608 /// </param>
609 /// <param name="argumentName">
610 /// <para>The argument name.</para>
611 /// <para></para>
612 /// </param>
613 /// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
614 /// <para></para>
615 /// <para></para>
616 /// </exception>
617 [MethodImpl(MethodImplOptions.AggressiveInlining)]
618 public static void EnsureInnerReferenceExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↳ links, TLinkAddress reference, string argumentName)

```



```

619 {
620     if (links.Constants.IsInternalReference(reference) && !links.Exists(reference))
621     {
622         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(reference,
623             ↪ argumentName);
624     }
625 }
626
627 /// <summary>
628 /// <para>
629 /// Ensures the inner reference exists using the specified links.
630 /// </para>
631 /// <para></para>
632 /// </summary>
633 /// <typeparam name="TLinkAddress">
634 /// <para>The link.</para>
635 /// <para></para>
636 /// </typeparam>
637 /// <param name="links">
638 /// <para>The links.</para>
639 /// <para></para>
640 /// </param>
641 /// <param name="restriction">
642 /// <para>The restriction.</para>
643 /// <para></para>
644 /// </param>
645 /// <param name="argumentName">
646 /// <para>The argument name.</para>
647 /// <para></para>
648 /// </param>
649 [MethodImpl(MethodImplOptions.AggressiveInlining)]
650 public static void EnsureInnerReferenceExists<TLinkAddress>(this ILinks<TLinkAddress>
651     ↪ links, IList<TLinkAddress>? restriction, string argumentName)
652 {
653     for (int i = 0; i < restriction.Count; i++)
654     {
655         links.EnsureInnerReferenceExists(restriction[i], argumentName);
656     }
657 }
658
659 /// <summary>
660 /// <para>
661 /// Ensures the link is any or exists using the specified links.
662 /// </para>
663 /// <para></para>
664 /// </summary>
665 /// <typeparam name="TLinkAddress">
666 /// <para>The link.</para>
667 /// <para></para>
668 /// </typeparam>
669 /// <param name="links">
670 /// <para>The links.</para>
671 /// <para></para>
672 /// </param>
673 /// <param name="restriction">
674 /// <para>The restriction.</para>
675 /// <para></para>
676 /// </param>
677 /// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
678 /// <para>sequence[{i}]</para>
679 /// <para></para>
680 /// </exception>
681 [MethodImpl(MethodImplOptions.AggressiveInlining)]
682 public static void EnsureLinkIsAnyOrExists<TLinkAddress>(this ILinks<TLinkAddress>
683     ↪ links, IList<TLinkAddress>? restriction)
684 {
685     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
686     var any = links.Constants.Any;
687     for (var i = 0; i < restriction.Count; i++)
688     {
689         if (!equalityComparer.Equals(restriction[i], any) &&
690             ↪ !links.Exists(restriction[i]))
691         {
692             throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(restriction[i],
693                 ↪ $"sequence[{i}]");
694         }
695     }
696 }

```

```

691 }
692
693 /// <summary>
694 /// <para>
695 /// Ensures the link is any or exists using the specified links.
696 /// </para>
697 /// <para></para>
698 /// </summary>
699 /// <typeparam name="TLinkAddress">
700 /// <para>The link.</para>
701 /// <para></para>
702 /// </typeparam>
703 /// <param name="links">
704 /// <para>The links.</para>
705 /// <para></para>
706 /// </param>
707 /// <param name="link">
708 /// <para>The link.</para>
709 /// <para></para>
710 /// </param>
711 /// <param name="argumentName">
712 /// <para>The argument name.</para>
713 /// <para></para>
714 /// </param>
715 /// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
716 /// <para></para>
717 /// <para></para>
718 /// </exception>
719 [MethodImpl(MethodImplOptions.AggressiveInlining)]
720 public static void EnsureLinkIsAnyOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, TLinkAddress link, string argumentName)
721 {
722     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
723     if (!equalityComparer.Equals(link, links.Constants.Any) && !links.Exists(link))
724     {
725         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
726     }
727 }
728
729 /// <summary>
730 /// <para>
731 /// Ensures the link is itself or exists using the specified links.
732 /// </para>
733 /// <para></para>
734 /// </summary>
735 /// <typeparam name="TLinkAddress">
736 /// <para>The link.</para>
737 /// <para></para>
738 /// </typeparam>
739 /// <param name="links">
740 /// <para>The links.</para>
741 /// <para></para>
742 /// </param>
743 /// <param name="link">
744 /// <para>The link.</para>
745 /// <para></para>
746 /// </param>
747 /// <param name="argumentName">
748 /// <para>The argument name.</para>
749 /// <para></para>
750 /// </param>
751 /// <exception cref="ArgumentLinkDoesNotExistsException{TLinkAddress}">
752 /// <para></para>
753 /// <para></para>
754 /// </exception>
755 [MethodImpl(MethodImplOptions.AggressiveInlining)]
756 public static void EnsureLinkIsItselfOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, TLinkAddress link, string argumentName)
757 {
758     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
759     if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))
760     {
761         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
762     }
763 }
764
765 /// <param name="links">Хранилище связей.</param>
766 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

767 public static void EnsureDoesNotExists<TLinkAddress>(this ILinks<TLinkAddress> links,
768     ↪ TLinkAddress source, TLinkAddress target)
769 {
770     if (links.Exists(source, target))
771     {
772         throw new LinkWithSameValueAlreadyExistsException();
773     }
774 }
775
776 /// <param name="links">Хранилище связей.</param>
777 [MethodImpl(MethodImplOptions.AggressiveInlining)]
778 public static void EnsureNoUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
779     ↪ TLinkAddress link)
780 {
781     if (links.HasUsages(link))
782     {
783         throw new ArgumentLinkHasDependenciesException<TLinkAddress>(link);
784     }
785 }
786
787 /// <param name="links">Хранилище связей.</param>
788 [MethodImpl(MethodImplOptions.AggressiveInlining)]
789 public static void EnsureCreated<TLinkAddress>(this ILinks<TLinkAddress> links, params
790     ↪ TLinkAddress[] addresses) => links.EnsureCreated(links.Create, addresses);
791
792 /// <param name="links">Хранилище связей.</param>
793 [MethodImpl(MethodImplOptions.AggressiveInlining)]
794 public static void EnsurePointsCreated<TLinkAddress>(this ILinks<TLinkAddress> links,
795     ↪ params TLinkAddress[] addresses) => links.EnsureCreated(links.CreatePoint,
796     ↪ addresses);
797
798 /// <param name="links">Хранилище связей.</param>
799 [MethodImpl(MethodImplOptions.AggressiveInlining)]
800 public static void EnsureCreated<TLinkAddress>(this ILinks<TLinkAddress> links,
801     ↪ Func<TLinkAddress> creator, params TLinkAddress[] addresses)
802 {
803     var addressToUInt64Converter = CheckedConverter<TLinkAddress, ulong>.Default;
804     var uInt64ToAddressConverter = CheckedConverter<ulong, TLinkAddress>.Default;
805     var nonExistentAddresses = new HashSet<TLinkAddress>(addresses.Where(x =>
806     ↪ !links.Exists(x)));
807     if (nonExistentAddresses.Count > 0)
808     {
809         var max = nonExistentAddresses.Max();
810         max = uInt64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
811     ↪ Convert(max),
812     ↪ addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
813     ↪ imum)));
814         var createdLinks = new List<TLinkAddress>();
815         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
816         TLinkAddress createdLink = creator();
817         while (!equalityComparer.Equals(createdLink, max))
818         {
819             createdLinks.Add(createdLink);
820         }
821         for (var i = 0; i < createdLinks.Count; i++)
822         {
823             if (!nonExistentAddresses.Contains(createdLinks[i]))
824             {
825                 links.Delete(createdLinks[i]);
826             }
827         }
828     }
829 }
830
831 #endregion
832
833 /// <param name="links">Хранилище связей.</param>
834 [MethodImpl(MethodImplOptions.AggressiveInlining)]
835 public static TLinkAddress CountUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
836     ↪ TLinkAddress link)
837 {
838     var constants = links.Constants;
839     var values = links.GetLink(link);
840     TLinkAddress usagesAsSource = links.Count(new Link<TLinkAddress>(constants.Any,
841     ↪ link, constants.Any));
842     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
843     if (equalityComparer.Equals(links.GetSource(values), link))
844     {
845

```

```

833     usagesAsSource = Arithmetic<TLinkAddress>.Decrement(usagesAsSource);
834 }
835 TLinkAddress usagesAsTarget = links.Count(new Link<TLinkAddress>(constants.Any,
    ↪ constants.Any, link));
836 if (equalityComparer.Equals(links.GetTarget(values), link))
837 {
838     usagesAsTarget = Arithmetic<TLinkAddress>.Decrement(usagesAsTarget);
839 }
840 return Arithmetic<TLinkAddress>.Add(usagesAsSource, usagesAsTarget);
841 }
842
843 /// <param name="links">Хранилище связей.</param>
844 [MethodImpl(MethodImplOptions.AggressiveInlining)]
845 public static bool HasUsages<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
    ↪ link) => Comparer<TLinkAddress>.Default.Compare(links.CountUsages(link), default) >
    ↪ 0;
846
847 /// <param name="links">Хранилище связей.</param>
848 [MethodImpl(MethodImplOptions.AggressiveInlining)]
849 public static bool Equals<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
    ↪ link, TLinkAddress source, TLinkAddress target)
850 {
851     var constants = links.Constants;
852     var values = links.GetLink(link);
853     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
854     return equalityComparer.Equals(links.GetSource(values), source) &&
        ↪ equalityComparer.Equals(links.GetTarget(values), target);
855 }
856
857 /// <summary>
858 /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
859 /// </summary>
860 /// <param name="links">Хранилище связей.</param>
861 /// <param name="source">Индекс связи, которая является началом для искомой
    ↪ связи.</param>
862 /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
863 /// <returns>Индекс искомой связи с указанными Source (началом) и Target
    ↪ (концом).</returns>
864 [MethodImpl(MethodImplOptions.AggressiveInlining)]
865 public static TLinkAddress SearchOrDefault<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, TLinkAddress source, TLinkAddress target)
866 {
867     var constants = links.Constants;
868     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
        ↪ constants.Break, default);
869     links.Each(setter.SetFirstAndReturnFalse, constants.Any, source, target);
870     return setter.Result;
871 }
872
873 public static TLinkAddress CreatePoint<TLinkAddress>(this ILinks<TLinkAddress> links)
874 {
875     var constants = links.Constants;
876     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
        ↪ constants.Break);
877     links.CreatePoint(setter.SetFirstFromSecondListAndReturnTrue);
878     return setter.Result;
879 }
880
881 /// <param name="links">Хранилище связей.</param>
882 [MethodImpl(MethodImplOptions.AggressiveInlining)]
883 public static TLinkAddress CreatePoint<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ WriteHandler<TLinkAddress>? handler)
884 {
885     var constants = links.Constants;
886     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
        ↪ constants.Break, handler);
887     TLinkAddress link = default;
888     TLinkAddress HandlerWrapper(ICollection<TLinkAddress>? before, ICollection<TLinkAddress>? after)
889     {
890         link = links.GetIndex(after);
891         return handlerState.Handle(before, after);
892     }
893     handlerState.Apply(links.Create(null, HandlerWrapper));
894     handlerState.Apply(links.Update(link, link, link, HandlerWrapper));
895     return handlerState.Result;
896 }
897

```

```

898 public static TLinkAddress CreateAndUpdate<TLinkAddress>(this ILinks<TLinkAddress>
899     ↳ links, TLinkAddress source, TLinkAddress target)
900 {
901     var constants = links.Constants;
902     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
903         ↳ constants.Break);
904     links.CreateAndUpdate(source, target, setter.SetFirstFromSecondListAndReturnTrue);
905     return setter.Result;
906 }
907
908 /// <param name="links">Хранилище связей.</param>
909 [MethodImpl(MethodImplOptions.AggressiveInlining)]
910 public static TLinkAddress CreateAndUpdate<TLinkAddress>(this ILinks<TLinkAddress>
911     ↳ links, TLinkAddress source, TLinkAddress target, WriteHandler<TLinkAddress>? handler)
912 {
913     var constants = links.Constants;
914     TLinkAddress createdLink = default;
915     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
916         ↳ constants.Break, handler);
917     handlerState.Apply(links.Create(null, (before, after) =>
918     {
919         createdLink = links.GetIndex(after);
920         return handlerState.Handle(before, after);
921     }));
922     handlerState.Apply(links.Update(createdLink, source, target, handler));
923     return handlerState.Result;
924 }
925
926 /// <summary>
927 /// Обновляет связь с указанными началом (Source) и концом (Target)
928 /// на связь с указанными началом (NewSource) и концом (NewTarget).
929 /// </summary>
930 /// <param name="links">Хранилище связей.</param>
931 /// <param name="link">Индекс обновляемой связи.</param>
932 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
933     ↳ выполняется обновление.</param>
934 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
935     ↳ выполняется обновление.</param>
936 /// <returns>Индекс обновлённой связи.</returns>
937 [MethodImpl(MethodImplOptions.AggressiveInlining)]
938 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
939     ↳ TLinkAddress link, TLinkAddress newSource, TLinkAddress newTarget) =>
940     links.Update(new LinkAddress<TLinkAddress>(link), new Link<TLinkAddress>(link,
941     ↳ newSource, newTarget));
942
943 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links, params
944     ↳ TLinkAddress[] restriction) => links.Update(restriction, null);
945
946 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
947     ↳ WriteHandler<TLinkAddress>? handler, params TLinkAddress[] restriction) =>
948     links.Update(restriction, handler);
949
950 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
951     ↳ IList<TLinkAddress>? restriction)
952 {
953     var constants = links.Constants;
954     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
955         ↳ constants.Break);
956     links.Update(restriction, setter.SetFirstFromSecondListAndReturnTrue);
957     return setter.Result;
958 }
959
960 /// <summary>
961 /// Обновляет связь с указанными началом (Source) и концом (Target)
962 /// на связь с указанными началом (NewSource) и концом (NewTarget).
963 /// </summary>
964 /// <param name="links">Хранилище связей.</param>
965 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
966     ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Itself -
967     ↳ требование установить ссылку на себя, 1..∞ конкретный адрес другой связи.</param>
968 /// <returns>Индекс обновлённой связи.</returns>
969 [MethodImpl(MethodImplOptions.AggressiveInlining)]
970 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
971     ↳ IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>? handler)
972 {
973     return restriction.Count switch

```

```

959     {
960         2 => links.MergeAndDelete(restriction[0], restriction[1], handler),
961         4 => links.UpdateOrCreateOrGet(restriction[0], restriction[1], restriction[2],
          ↳ restriction[3], handler),
962         _ => links.Update(restriction[0], restriction[1], restriction[2], handler)
963     };
964 }
965
966 public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
          ↳ TLinkAddress link, TLinkAddress newSource, TLinkAddress newTarget,
          ↳ WriteHandler<TLinkAddress>? handler) => links.Update(new
          ↳ LinkAddress<TLinkAddress>(link), new Link<TLinkAddress>(link, newSource, newTarget),
          ↳ handler);
967
968 /// <summary>
969 /// <para>
970 /// Resolves the constant as self reference using the specified links.
971 /// </para>
972 /// <para></para>
973 /// </summary>
974 /// <typeparam name="TLinkAddress">
975 /// <para>The link.</para>
976 /// <para></para>
977 /// </typeparam>
978 /// <param name="links">
979 /// <para>The links.</para>
980 /// <para></para>
981 /// </param>
982 /// <param name="constant">
983 /// <para>The constant.</para>
984 /// <para></para>
985 /// </param>
986 /// <param name="restriction">
987 /// <para>The restriction.</para>
988 /// <para></para>
989 /// </param>
990 /// <param name="substitution">
991 /// <para>The substitution.</para>
992 /// <para></para>
993 /// </param>
994 /// <returns>
995 /// <para>A list of t link</para>
996 /// <para></para>
997 /// </returns>
998 [MethodImpl(MethodImplOptions.AggressiveInlining)]
999 public static IList<TLinkAddress>? ResolveConstantAsSelfReference<TLinkAddress>(this
          ↳ ILinks<TLinkAddress> links, TLinkAddress constant, IList<TLinkAddress>? restriction,
          ↳ IList<TLinkAddress>? substitution)
1000 {
1001     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1002     var constants = links.Constants;
1003     var restrictionIndex = links.GetIndex(restriction);
1004     var substitutionIndex = links.GetIndex(substitution);
1005     if (equalityComparer.Equals(substitutionIndex, default))
1006     {
1007         substitutionIndex = restrictionIndex;
1008     }
1009     var source = links.GetSource(substitution);
1010     var target = links.GetTarget(substitution);
1011     source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;
1012     target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
1013     return new Link<TLinkAddress>(substitutionIndex, source, target);
1014 }
1015
1016 /// <summary>
1017 /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
          ↳ с указанными Source (началом) и Target (концом).
1018 /// </summary>
1019 /// <param name="links">Хранилище связей.</param>
1020 /// <param name="source">Индекс связи, которая является началом на создаваемой
          ↳ связи.</param>
1021 /// <param name="target">Индекс связи, которая является концом для создаваемой
          ↳ связи.</param>
1022 /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
1023 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1024 public static TLinkAddress GetOrCreate<TLinkAddress>(this ILinks<TLinkAddress> links,
          ↳ TLinkAddress source, TLinkAddress target)

```

```

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

```

```

1091     for (int i = 0; i < deletedLinks.Count; i++)
1092     {
1093         links.Delete(deletedLinks[i]);
1094     }
1095 }
1096
1097 public static void DeleteAllUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
1098     ↪ TLinkAddress linkIndex) => links.DeleteAllUsages(linkIndex, null);
1099
1100 /// <remarks>Before execution of this method ensure that deleted link is detached (all
1101 ↪ values - source and target are reset to null) or it might enter into infinite
1102 ↪ recursion.</remarks>
1103 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1104 public static TLinkAddress DeleteAllUsages<TLinkAddress>(this ILinks<TLinkAddress>
1105     ↪ links, TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1106 {
1107     var constants = links.Constants;
1108     var any = constants.Any;
1109     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1110     var usagesAsSourceQuery = new Link<TLinkAddress>(any, linkIndex, any);
1111     var usagesAsTargetQuery = new Link<TLinkAddress>(any, any, linkIndex);
1112     var usages = new List<IList<TLinkAddress>?>();
1113     var usagesFiller = new ListFiller<IList<TLinkAddress>?, TLinkAddress>(usages,
1114         ↪ constants.Continue);
1115     links.Each(usagesFiller.AddAndReturnConstant, usagesAsSourceQuery);
1116     links.Each(usagesFiller.AddAndReturnConstant, usagesAsTargetQuery);
1117     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
1118         ↪ constants.Break, handler);
1119     foreach (var usage in usages)
1120     {
1121         if (equalityComparer.Equals(links.GetIndex(usage), linkIndex) ||
1122             ↪ !links.Exists(links.GetIndex(usage)))
1123         {
1124             continue;
1125         }
1126         handlerState.Apply(links.Delete(links.GetIndex(usage), handlerState.Handler));
1127     }
1128     return handlerState.Result;
1129 }
1130
1131 /// <summary>
1132 /// <para>
1133 /// Deletes the by query using the specified links.
1134 /// </para>
1135 /// <para></para>
1136 /// </summary>
1137 /// <typeparam name="TLinkAddress">
1138 /// <para>The link.</para>
1139 /// <para></para>
1140 /// </typeparam>
1141 /// <param name="links">
1142 /// <para>The links.</para>
1143 /// <para></para>
1144 /// </param>
1145 /// <param name="query">
1146 /// <para>The query.</para>
1147 /// <para></para>
1148 /// </param>
1149 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1150 public static void DeleteByQuery<TLinkAddress>(this ILinks<TLinkAddress> links,
1151     ↪ Link<TLinkAddress> query)
1152 {
1153     var queryResult = new List<TLinkAddress>();
1154     var queryResultFiller = new ListFiller<TLinkAddress, TLinkAddress>(queryResult,
1155         ↪ links.Constants.Continue);
1156     links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
1157     foreach (var link in queryResult)
1158     {
1159         links.Delete(link);
1160     }
1161 }
1162
1163 // TODO: Move to Platform.Data
1164 /// <summary>
1165 /// <para>
1166 /// Determines whether are values reset.
1167 /// </para>
1168 /// <para></para>

```



```

1160     /// </summary>
1161     /// <typeparam name="TLinkAddress">
1162     /// <para>The link.</para>
1163     /// <para></para>
1164     /// </typeparam>
1165     /// <param name="links">
1166     /// <para>The links.</para>
1167     /// <para></para>
1168     /// </param>
1169     /// <param name="linkIndex">
1170     /// <para>The link index.</para>
1171     /// <para></para>
1172     /// </param>
1173     /// <returns>
1174     /// <para>The bool</para>
1175     /// <para></para>
1176     /// </returns>
1177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1178     public static bool AreValuesReset<TLinkAddress>(this ILinks<TLinkAddress> links,
1179     ↪ TLinkAddress linkIndex)
1180     {
1181         var nullConstant = links.Constants.Null;
1182         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1183         var link = links.GetLink(linkIndex);
1184         for (int i = 1; i < link.Count; i++)
1185         {
1186             if (!equalityComparer.Equals(link[i], nullConstant))
1187             {
1188                 return false;
1189             }
1190         }
1191         return true;
1192     }
1193
1194     public static void ResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1195     ↪ TLinkAddress linkIndex) => links.ResetValues(linkIndex, null);
1196
1197     // TODO: Create a universal version of this method in Platform.Data (with using of for
1198     ↪ loop)
1199     /// <summary>
1200     /// <para>
1201     /// Resets the values using the specified links.
1202     /// </para>
1203     /// <para></para>
1204     /// </summary>
1205     /// <typeparam name="TLinkAddress">
1206     /// <para>The link.</para>
1207     /// <para></para>
1208     /// </typeparam>
1209     /// <param name="links">
1210     /// <para>The links.</para>
1211     /// <para></para>
1212     /// </param>
1213     /// <param name="linkIndex">
1214     /// <para>The link index.</para>
1215     /// <para></para>
1216     /// </param>
1217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1218     public static TLinkAddress ResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1219     ↪ TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1220     {
1221         var nullConstant = links.Constants.Null;
1222         var updateRequest = new Link<TLinkAddress>(linkIndex, nullConstant, nullConstant);
1223         return links.Update(updateRequest, handler);
1224     }
1225
1226     public static void EnforceResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1227     ↪ TLinkAddress linkIndex) => links.EnforceResetValues(linkIndex, null);
1228
1229     // TODO: Create a universal version of this method in Platform.Data (with using of for
1230     ↪ loop)
1231     /// <summary>
1232     /// <para>
1233     /// Enforces the reset values using the specified links.
1234     /// </para>
1235     /// <para></para>
1236     /// </summary>

```

```

1232    /// <typeparam name="TLinkAddress">
1233    /// <para>The link.</para>
1234    /// <para></para>
1235    /// </typeparam>
1236    /// <param name="links">
1237    /// <para>The links.</para>
1238    /// <para></para>
1239    /// </param>
1240    /// <param name="linkIndex">
1241    /// <para>The link index.</para>
1242    /// <para></para>
1243    /// </param>
1244    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1245    public static TLinkAddress EnforceResetValues<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1246    {
1247        if (!links.AreValuesReset(linkIndex))
1248        {
1249            return links.ResetValues(linkIndex, handler);
1250        }
1251        return links.Constants.Continue;
1252    }
1253
1254    public static void MergeUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex) =>
    ↪ links.MergeUsages(oldLinkIndex, newLinkIndex, null);
1255
1256    /// <summary>
1257    /// Merging two usages graphs, all children of old link moved to be children of new link
    ↪ or deleted.
1258    /// </summary>
1259    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1260    public static TLinkAddress MergeUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex, WriteHandler<TLinkAddress>?
    ↪ handler)
1261    {
1262        var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1263        if (equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1264        {
1265            return newLinkIndex;
1266        }
1267        var constants = links.Constants;
1268        var usagesAsSource = links.All(new Link<TLinkAddress>(constants.Any, oldLinkIndex,
    ↪ constants.Any));
1269        WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
    ↪ constants.Break, handler);
1270        for (var i = 0; i < usagesAsSource.Count; i++)
1271        {
1272            var usageAsSource = usagesAsSource[i];
1273            if (equalityComparer.Equals(links.GetIndex(usageAsSource), oldLinkIndex))
1274            {
1275                continue;
1276            }
1277            var restriction = new LinkAddress<TLinkAddress>(links.GetIndex(usageAsSource));
1278            var substitution = new Link<TLinkAddress>(newLinkIndex,
    ↪ links.GetTarget(usageAsSource));
1279            handlerState.Apply(links.Update(restriction, substitution,
    ↪ handlerState.Handler));
1280        }
1281        var usagesAsTarget = links.All(new Link<TLinkAddress>(constants.Any, constants.Any,
    ↪ oldLinkIndex));
1282        for (var i = 0; i < usagesAsTarget.Count; i++)
1283        {
1284            var usageAsTarget = usagesAsTarget[i];
1285            if (equalityComparer.Equals(links.GetIndex(usageAsTarget), oldLinkIndex))
1286            {
1287                continue;
1288            }
1289            var restriction = links.GetLink(links.GetIndex(usageAsTarget));
1290            var substitution = new Link<TLinkAddress>(links.GetTarget(usageAsTarget),
    ↪ newLinkIndex);
1291            handlerState.Apply(links.Update(restriction, substitution,
    ↪ handlerState.Handler));
1292        }
1293        return handlerState.Result;
1294    }
1295

```

```

1296 public static TLinkAddress MergeAndDelete<TLinkAddress>(this ILinks<TLinkAddress> links,
1297     ↳ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex)
1298 {
1299     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1300     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1301     {
1302         links.MergeUsages(oldLinkIndex, newLinkIndex);
1303         links.Delete(oldLinkIndex);
1304     }
1305     return newLinkIndex;
1306 }
1307
1308 /// <summary>
1309 /// Replace one link with another (replaced link is deleted, children are updated or
1310 /// ↳ deleted).
1311 /// </summary>
1312 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1313 public static TLinkAddress MergeAndDelete<TLinkAddress>(this ILinks<TLinkAddress> links,
1314     ↳ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex, WriteHandler<TLinkAddress>?
1315     ↳ handler)
1316 {
1317     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1318     var constants = links.Constants;
1319     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
1320     ↳ constants.Break, handler);
1321     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1322     {
1323         handlerState.Apply(links.MergeUsages(oldLinkIndex, newLinkIndex,
1324         ↳ handlerState.Handler));
1325         handlerState.Apply(links.Delete(oldLinkIndex, handlerState.Handler));
1326     }
1327     return handlerState.Result;
1328 }
1329
1330 /// <summary>
1331 /// <para>
1332 /// Decorates the with automatic uniqueness and usages resolution using the specified
1333 /// ↳ links.
1334 /// </para>
1335 /// <para></para>
1336 /// </summary>
1337 /// <typeparam name="TLinkAddress">
1338 /// <para>The link.</para>
1339 /// <para></para>
1340 /// </typeparam>
1341 /// <param name="links">
1342 /// <para>The links.</para>
1343 /// <para></para>
1344 /// </param>
1345 /// <returns>
1346 /// <para>The links.</para>
1347 /// <para></para>
1348 /// </returns>
1349 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1350 public static ILinks<TLinkAddress>
1351     ↳ DecorateWithAutomaticUniquenessAndUsagesResolution<TLinkAddress>(this
1352     ↳ ILinks<TLinkAddress> links)
1353 {
1354     links = new LinksCascadeUsagesResolver<TLinkAddress>(links);
1355     links = new NonNullContentsLinkDeletionResolver<TLinkAddress>(links);
1356     links = new LinksCascadeUniquenessAndUsagesResolver<TLinkAddress>(links);
1357     return links;
1358 }
1359
1360 /// <summary>
1361 /// <para>
1362 /// Formats the links.
1363 /// </para>
1364 /// <para></para>
1365 /// </summary>
1366 /// <typeparam name="TLinkAddress">
1367 /// <para>The link.</para>
1368 /// <para></para>
1369 /// </typeparam>
1370 /// <param name="links">
1371 /// <para>The links.</para>
1372 /// <para></para>
1373 /// </param>

```

```

1365     /// <param name="link">
1366     /// <para>The link.</para>
1367     /// <para></para>
1368     /// </param>
1369     /// <returns>
1370     /// <para>The string</para>
1371     /// <para></para>
1372     /// </returns>
1373     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1374     public static string Format<TLinkAddress>(this ILinks<TLinkAddress> links,
1375     ↪ IList<TLinkAddress>? link)
1376     {
1377         var constants = links.Constants;
1378         return $"({links.GetIndex(link)}: {links.GetSource(link)} {links.GetTarget(link)})";
1379     }
1380     /// <summary>
1381     /// <para>
1382     /// Formats the links.
1383     /// </para>
1384     /// <para></para>
1385     /// </summary>
1386     /// <typeparam name="TLinkAddress">
1387     /// <para>The link.</para>
1388     /// <para></para>
1389     /// </typeparam>
1390     /// <param name="links">
1391     /// <para>The links.</para>
1392     /// <para></para>
1393     /// </param>
1394     /// <param name="link">
1395     /// <para>The link.</para>
1396     /// <para></para>
1397     /// </param>
1398     /// <returns>
1399     /// <para>The string</para>
1400     /// <para></para>
1401     /// </returns>
1402     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1403     public static string Format<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
1404     ↪ link) => links.Format(links.GetLink(link));
1405 }

```

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

```

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

```

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

```

1  using Platform.Collections.Lists;
2  using Platform.Exceptions;
3  using Platform.Ranges;
4  using Platform.Singletons;
5  using System;
6  using System.Collections;
7  using System.Collections.Generic;
8  using System.Runtime.CompilerServices;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets
13 {

```

```

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

```

```

87     /// </param>
88     /// <exception cref="NotSupportedException">
89     /// <para></para>
90     /// <para></para>
91     /// </exception>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public Link(object other)
94     {
95         if (other is Link<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     /// <para></para>
114     /// </summary>
115     /// <param name="other">
116     /// <para>A other.</para>
117     /// <para></para>
118     /// </param>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     public Link(ref Link<TLinkAddress> other) => SetValues(ref other, out Index, out Source,
121     ↪ out Target);
122
123     /// <summary>
124     /// <para>
125     /// Initializes a new <see cref="Link"/> instance.
126     /// </para>
127     /// <para></para>
128     /// </summary>
129     /// <param name="index">
130     /// <para>A index.</para>
131     /// <para></para>
132     /// </param>
133     /// <param name="source">
134     /// <para>A source.</para>
135     /// <para></para>
136     /// </param>
137     /// <param name="target">
138     /// <para>A target.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public Link(TLinkAddress index, TLinkAddress source, TLinkAddress target)
143     {
144         Index = index;
145         Source = source;
146         Target = target;
147     }
148     [MethodImpl(MethodImplOptions.AggressiveInlining)]
149     private static void SetValues(ref Link<TLinkAddress> other, out TLinkAddress index, out
150     ↪ TLinkAddress source, out TLinkAddress target)
151     {
152         index = other.Index;
153         source = other.Source;
154         target = other.Target;
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         {
162             index = default;
163             source = default;
164             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> &&
233     ↪ Equals((Link<TLinkAddress>)other);
234
235 /// <summary>
236 /// <para>
237 /// Determines whether this instance equals.
238 /// </para>
239 /// <para></para>
240 /// </summary>

```

```

240    /// <param name="other">
241    /// <para>The other.</para>
242    /// <para></para>
243    /// </param>
244    /// <returns>
245    /// <para>The bool</para>
246    /// <para></para>
247    /// </returns>
248    [MethodImpl(MethodImplOptions.AggressiveInlining)]
249    public bool Equals(Link<TLinkAddress> other) => _equalityComparer.Equals(Index,
    ↪ other.Index)
    && _equalityComparer.Equals(Source, other.Source)
    && _equalityComparer.Equals(Target, other.Target);

250
251
252
253    /// <summary>
254    /// <para>
255    /// Returns the string using the specified index.
256    /// </para>
257    /// <para></para>
258    /// </summary>
259    /// <param name="index">
260    /// <para>The index.</para>
261    /// <para></para>
262    /// </param>
263    /// <param name="source">
264    /// <para>The source.</para>
265    /// <para></para>
266    /// </param>
267    /// <param name="target">
268    /// <para>The target.</para>
269    /// <para></para>
270    /// </param>
271    /// <returns>
272    /// <para>The string</para>
273    /// <para></para>
274    /// </returns>
275    [MethodImpl(MethodImplOptions.AggressiveInlining)]
276    public static string ToString(TLinkAddress index, TLinkAddress source, TLinkAddress
    ↪ target) => $"{index}: {source}->{target}";

277
278    /// <summary>
279    /// <para>
280    /// Returns the string using the specified source.
281    /// </para>
282    /// <para></para>
283    /// </summary>
284    /// <param name="source">
285    /// <para>The source.</para>
286    /// <para></para>
287    /// </param>
288    /// <param name="target">
289    /// <para>The target.</para>
290    /// <para></para>
291    /// </param>
292    /// <returns>
293    /// <para>The string</para>
294    /// <para></para>
295    /// </returns>
296    [MethodImpl(MethodImplOptions.AggressiveInlining)]
297    public static string ToString(TLinkAddress source, TLinkAddress target) =>
    ↪ $"{source}->{target}";

298
299    [MethodImpl(MethodImplOptions.AggressiveInlining)]
300    public static implicit operator TLinkAddress[] (Link<TLinkAddress> link) =>
    ↪ link.ToArray();

301
302    [MethodImpl(MethodImplOptions.AggressiveInlining)]
303    public static implicit operator Link<TLinkAddress>(TLinkAddress[] linkArray) => new
    ↪ Link<TLinkAddress>(linkArray);

304
305    /// <summary>
306    /// <para>
307    /// Returns the string.
308    /// </para>
309    /// <para></para>
310    /// </summary>
311    /// <returns>

```



```

312     /// <para>The string</para>
313     /// <para></para>
314     /// </returns>
315     [MethodImpl(MethodImplOptions.AggressiveInlining)]
316     public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
        ↳ ToString(Source, Target) : ToString(Index, Source, Target);

317
318     #region IList
319
320     /// <summary>
321     /// <para>
322     /// Gets the count value.
323     /// </para>
324     /// <para></para>
325     /// </summary>
326     public int Count
327     {
328         [MethodImpl(MethodImplOptions.AggressiveInlining)]
329         get => Length;
330     }
331
332     /// <summary>
333     /// <para>
334     /// Gets the is read only value.
335     /// </para>
336     /// <para></para>
337     /// </summary>
338     public bool IsReadOnly
339     {
340         [MethodImpl(MethodImplOptions.AggressiveInlining)]
341         get => true;
342     }
343
344     /// <summary>
345     /// <para>
346     /// The not supported exception.
347     /// </para>
348     /// <para></para>
349     /// </summary>
350     public 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),
467         ↳ nameof(arrayIndex));
468     if (arrayIndex + Length > array.Length)
469     {
470         throw new InvalidOperationException();
471     }
472     array[arrayIndex++] = Index;
473     array[arrayIndex++] = Source;
474     array[arrayIndex] = Target;
475 }
476
477 /// <summary>
478 /// <para>
479 /// Determines whether this instance remove.
480 /// </para>
481 /// <para></para>
482 /// </summary>
483 /// <param name="item">
484 /// <para>The item.</para>
485 /// <para></para>
486 /// </param>
487 /// <returns>
488 /// <para>The bool</para>
489 /// <para></para>
490 /// </returns>
491 [MethodImpl(MethodImplOptions.AggressiveInlining)]
492 public bool Remove(TLinkAddress item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
493
494 /// <summary>
495 /// <para>
496 /// Indexes the of using the specified item.
497 /// </para>
498 /// <para></para>
499 /// </summary>
500 /// <param name="item">
501 /// <para>The item.</para>
502 /// <para></para>
503 /// </param>
504 /// <returns>
505 /// <para>The int</para>
506 /// <para></para>
507 /// </returns>
508 [MethodImpl(MethodImplOptions.AggressiveInlining)]
509 public int IndexOf(TLinkAddress item)
510 {
511     if (_equalityComparer.Equals(Index, item))
512     {
513         return _constants.IndexPart;
514     }
515     if (_equalityComparer.Equals(Source, item))
516     {
517         return _constants.SourcePart;
518     }
519     if (_equalityComparer.Equals(Target, item))
520     {
521         return _constants.TargetPart;
522     }
523     return -1;
524 }
525
526 /// <summary>
527 /// <para>
528 /// Inserts the index.
529 /// </para>
530 /// <para></para>
531 /// </summary>
532 /// <param name="index">
533 /// <para>The index.</para>
534 /// <para></para>
535 /// </param>
536 /// <param name="item">
537 /// <para>The item.</para>
538 /// <para></para>
539 /// </param>
540 [MethodImpl(MethodImplOptions.AggressiveInlining)]
541 public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
542
543 /// <summary>

```

```

543     /// <para>
544     /// Removes the at using the specified index.
545     /// </para>
546     /// <para></para>
547     /// </summary>
548     /// <param name="index">
549     /// <para>The index.</para>
550     /// <para></para>
551     /// </param>
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]
553     public void RemoveAt(int index) => throw new NotSupportedException();
554
555     [MethodImpl(MethodImplOptions.AggressiveInlining)]
556     public static bool operator ==(Link<TLinkAddress> left, Link<TLinkAddress> right) =>
557         left.Equals(right);
558
559     [MethodImpl(MethodImplOptions.AggressiveInlining)]
560     public static bool operator !=(Link<TLinkAddress> left, Link<TLinkAddress> right) =>
561         !(left == right);
562
563     #endregion
564 }
565 }

```

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

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links operator base.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public abstract class LinksOperatorBase<TLinkAddress>
14     {
15         /// <summary>
16         /// <para>
17         /// The links.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         protected readonly ILinks<TLinkAddress> _links;
22
23         /// <summary>
24         /// <para>
25         /// Gets the links value.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         public ILinks<TLinkAddress> Links
30         {
31             [MethodImpl(MethodImplOptions.AggressiveInlining)]
32             get => _links;
33         }
34
35         /// <summary>
36         /// <para>
37         /// Initializes a new <see cref="LinksOperatorBase"/> instance.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="links">
42         /// <para>A links.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         protected LinksOperatorBase(ILinks<TLinkAddress> links) => _links = links;
47     }
48 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory
6  {
7      /// <summary>
8      /// <para>
9      /// Defines the links list methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public interface ILinksListMethods<TLinkAddress>
14     {
15         /// <summary>
16         /// <para>
17         /// Detaches the free link.
18         /// </para>
19         /// <para></para>
20         /// </summary>

```

```

21     /// <param name="freeLink">
22     /// <para>The free link.</para>
23     /// <para></para>
24     /// </param>
25     [MethodImpl(MethodImplOptions.AggressiveInlining)]
26     void Detach(TLinkAddress freeLink);
27
28     /// <summary>
29     /// <para>
30     /// Attaches the as first using the specified link.
31     /// </para>
32     /// <para></para>
33     /// </summary>
34     /// <param name="link">
35     /// <para>The link.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     void AttachAsFirst(TLinkAddress link);
40 }
41 }

```

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

```

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

```

```

57     /// <para>
58     /// Eaches the usage using the specified root.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="root">
63     /// <para>The root.</para>
64     /// <para></para>
65     /// </param>
66     /// <param name="handler">
67     /// <para>The handler.</para>
68     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The link</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     TLinkAddress EachUsage(TLinkAddress root, ReadHandler<TLinkAddress>? handler);
76
77     /// <summary>
78     /// <para>
79     /// Detaches the root.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="root">
84     /// <para>The root.</para>
85     /// <para></para>
86     /// </param>
87     /// <param name="linkIndex">
88     /// <para>The link index.</para>
89     /// <para></para>
90     /// </param>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     void Detach(ref TLinkAddress root, TLinkAddress linkIndex);
93
94     /// <summary>
95     /// <para>
96     /// Attaches the root.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="root">
101    /// <para>The root.</para>
102    /// <para></para>
103    /// </param>
104    /// <param name="linkIndex">
105    /// <para>The link index.</para>
106    /// <para></para>
107    /// </param>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    void Attach(ref TLinkAddress root, TLinkAddress linkIndex);
110 }
111 }

```

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

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets.Memory
4  {
5      /// <summary>
6      /// <para>
7      /// The index tree type enum.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public enum IndexTreeType
12     {
13         /// <summary>
14         /// <para>
15         /// The default index tree type.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         Default = 0,
20         /// <summary>
21         /// <para>

```

```

22      /// The size balanced tree index tree type.
23      /// </para>
24      /// <para></para>
25      /// </summary>
26      SizeBalancedTree = 1,
27      /// <summary>
28      /// <para>
29      /// The recursionless size balanced tree index tree type.
30      /// </para>
31      /// <para></para>
32      /// </summary>
33      RecursionlessSizeBalancedTree = 2,
34      /// <summary>
35      /// <para>
36      /// The sized and threaded avl balanced tree index tree type.
37      /// </para>
38      /// <para></para>
39      /// </summary>
40      SizedAndThreadedAVLBalancedTree = 3
41  }
42  }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Unsafe;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Memory
9 {
10     /// <summary>
11     /// <para>
12     /// The links header.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct LinksHeader<TLinkAddress> : IEquatable<LinksHeader<TLinkAddress>>
17     {
18         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
19             EqualityComparer<TLinkAddress>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<LinksHeader<TLinkAddress>>.Size;
28
29         /// <summary>
30         /// <para>
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         /// <para></para>
58         /// </summary>
59         public TLinkAddress FirstFreeLink;
60     }
61 }

```



```

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.
131    /// </para>
132    /// <para></para>
133    /// </summary>

```

```

133     /// <returns>
134     /// <para>The int</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public override int GetHashCode() => (AllocatedLinks, ReservedLinks, FreeLinks,
        ↪ FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();
139
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     public static bool operator ==(LinksHeader<TLinkAddress> left, LinksHeader<TLinkAddress>
        ↪ right) => left.Equals(right);
142
143     [MethodImpl(MethodImplOptions.AggressiveInlining)]
144     public static bool operator !=(LinksHeader<TLinkAddress> left, LinksHeader<TLinkAddress>
        ↪ right) => !(left == right);
145 }
146 }

```

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

```

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

```

```

58     /// <para></para>
59     /// </summary>
60     protected readonly byte* Header;
61
62     /// <summary>
63     /// <para>
64     /// Initializes a new <see
        ↪ cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> 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
        ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(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.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <returns>
101    /// <para>The link</para>
102    /// <para></para>
103    /// </returns>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    protected abstract TLinkAddress GetTreeRoot();
106
107    /// <summary>
108    /// <para>
109    /// Gets the base part value using the specified link.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="link">
114    /// <para>The link.</para>
115    /// <para></para>
116    /// </param>
117    /// <returns>
118    /// <para>The link</para>
119    /// <para></para>
120    /// </returns>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    protected abstract TLinkAddress GetBasePartValue(TLinkAddress link);
123
124    /// <summary>
125    /// <para>
126    /// Determines whether this instance first is to the right of second.
127    /// </para>
128    /// <para></para>
129    /// </summary>
130    /// <param name="source">
131    /// <para>The source.</para>
132    /// <para></para>

```

```

133     /// </param>
134     /// <param name="target">
135     /// <para>The target.</para>
136     /// <para></para>
137     /// </param>
138     /// <param name="rootSource">
139     /// <para>The root source.</para>
140     /// <para></para>
141     /// </param>
142     /// <param name="rootTarget">
143     /// <para>The root target.</para>
144     /// <para></para>
145     /// </param>
146     /// <returns>
147     /// <para>The bool</para>
148     /// <para></para>
149     /// </returns>
150     [MethodImpl(MethodImplOptions.AggressiveInlining)]
151     protected abstract bool FirstIsToTheRightOfSecond(TLinkAddress source, TLinkAddress
        ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);

152     /// <summary>
153     /// <para>
154     /// 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     /// 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     /// 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 /// </returns>
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 protected virtual ref RawLinkDataPart<TLinkAddress>
211 ↪ GetLinkDataPartReference(TLinkAddress link) => ref
212 ↪ AsRef<RawLinkDataPart<TLinkAddress>>(LinksDataParts +
213 ↪ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
214 ↪ _addressToInt64Converter.Convert(link)));
215
216 /// <summary>
217 /// <para>
218 /// Gets the link index part reference using the specified link.
219 /// </para>
220 /// <para></para>
221 /// </summary>
222 /// <param name="link">
223 /// <para>The link.</para>
224 /// <para></para>
225 /// </param>
226 /// <returns>
227 /// <para>A ref raw link index part of t link</para>
228 /// <para></para>
229 /// </returns>
230 [MethodImpl(MethodImplOptions.AggressiveInlining)]
231 protected virtual ref RawLinkIndexPart<TLinkAddress>
232 ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
233 ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
234 ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
235 ↪ _addressToInt64Converter.Convert(link)));
236
237 /// <summary>
238 /// <para>
239 /// Gets the link values 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 list of t link</para>
249 /// <para></para>
250 /// </returns>
251 [MethodImpl(MethodImplOptions.AggressiveInlining)]
252 protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
253 {
254     ref var link = ref GetLinkDataPartReference(linkIndex);
255     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
256 }
257
258 /// <summary>
259 /// <para>
260 /// Determines whether this instance first is to the left of second.
261 /// </para>
262 /// <para></para>
263 /// </summary>
264 /// <param name="first">
265 /// <para>The first.</para>
266 /// <para></para>
267 /// </param>
268 /// <param name="second">
269 /// <para>The second.</para>
270 /// <para></para>
271 /// </param>
272 /// <returns>
273 /// <para>The bool</para>
274 /// <para></para>
275 /// </returns>
276 [MethodImpl(MethodImplOptions.AggressiveInlining)]
277 protected override bool FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
278 {
279     ref var firstLink = ref GetLinkDataPartReference(first);
280     ref var secondLink = ref GetLinkDataPartReference(second);
281     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
282 ↪ secondLink.Source, secondLink.Target);
283 }
284
285 /// <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 // 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 /// <summary>
422 /// <para>
423 /// Eaches the usage using the specified base.
424 /// </para>
425 /// <para></para>
426 /// </summary>

```

```

427     /// <param name="@base">
428     /// <para>The base.</para>
429     /// <para></para>
430     /// </param>
431     /// <param name="handler">
432     /// <para>The handler.</para>
433     /// <para></para>
434     /// </param>
435     /// <returns>
436     /// <para>The link</para>
437     /// <para></para>
438     /// </returns>
439     [MethodImpl(MethodImplOptions.AggressiveInlining)]
440     public TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
441         ↳ EachUsageCore(@base, GetTreeRoot(), handler);
442
443     // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
444     ↳ low-level MSIL stack.
445     [MethodImpl(MethodImplOptions.AggressiveInlining)]
446     private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
447         ↳ ReadHandler<TLinkAddress>? handler)
448     {
449         var @continue = Continue;
450         if (EqualToZero(link))
451         {
452             return @continue;
453         }
454         var linkBasePart = GetBasePartValue(link);
455         var @break = Break;
456         if (GreaterThan(linkBasePart, @base))
457         {
458             if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
459             {
460                 return @break;
461             }
462         }
463         else if (LessThan(linkBasePart, @base))
464         {
465             if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
466             {
467                 return @break;
468             }
469         }
470         else //if (linkBasePart == @base)
471         {
472             if (AreEqual(handler(GetLinkValues(link)), @break))
473             {
474                 return @break;
475             }
476             if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
477             {
478                 return @break;
479             }
480             if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
481             {
482                 return @break;
483             }
484         }
485         return @continue;
486     }
487
488     /// <summary>
489     /// <para>
490     /// Prints the node value using the specified node.
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     {

```



```

502         ref var link = ref GetLinkDataPartReference(node);
503         sb.Append(' ');
504         sb.Append(link.Source);
505         sb.Append('-');
506         sb.Append('>');
507         sb.Append(link.Target);
508     }
509 }
510 }

```

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

```

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

```

```

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

```

```

144    /// <para></para>
145    /// </param>
146    /// <returns>
147    /// <para>The bool</para>
148    /// <para></para>
149    /// </returns>
150    [MethodImpl(MethodImplOptions.AggressiveInlining)]
151    protected abstract bool FirstIsToTheRightOfSecond(TLinkAddress source, TLinkAddress
        ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);

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

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

194
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)) //
        ↪ node.Key < root.Key
354             {
355                 root = GetLeftOrDefault(root);
356             }
357             else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
        ↪ node.Key > root.Key
358             {
359                 root = GetRightOrDefault(root);

```

```

360     }
361     else // node.Key == root.Key
362     {
363         return root;
364     }
365 }
366 return Zero;
367 }
368
369 // TODO: Return indices range instead of references count
370 /// <summary>
371 /// <para>
372 /// Counts the usages using the specified link.
373 /// </para>
374 /// <para></para>
375 /// </summary>
376 /// <param name="link">
377 /// <para>The link.</para>
378 /// <para></para>
379 /// </param>
380 /// <returns>
381 /// <para>The link</para>
382 /// <para></para>
383 /// </returns>
384 [MethodImpl(MethodImplOptions.AggressiveInlining)]
385 public TLinkAddress CountUsages(TLinkAddress link)
386 {
387     var root = GetTreeRoot();
388     var total = GetSize(root);
389     var totalRightIgnore = Zero;
390     while (!EqualToZero(root))
391     {
392         var @base = GetBasePartValue(root);
393         if (LessOrEqualThan(@base, link))
394         {
395             root = GetRightOrDefault(root);
396         }
397         else
398         {
399             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
400             root = GetLeftOrDefault(root);
401         }
402     }
403     root = GetTreeRoot();
404     var totalLeftIgnore = Zero;
405     while (!EqualToZero(root))
406     {
407         var @base = GetBasePartValue(root);
408         if (GreaterOrEqualThan(@base, link))
409         {
410             root = GetLeftOrDefault(root);
411         }
412         else
413         {
414             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
415             root = GetRightOrDefault(root);
416         }
417     }
418     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
419 }
420
421 /// <summary>
422 /// <para>
423 /// Eaches the usage using the specified base.
424 /// </para>
425 /// <para></para>
426 /// </summary>
427 /// <param name="@base">
428 /// <para>The base.</para>
429 /// <para></para>
430 /// </param>
431 /// <param name="handler">
432 /// <para>The handler.</para>
433 /// <para></para>
434 /// </param>
435 /// <returns>
436 /// <para>The link</para>
437 /// <para></para>

```

```

438 /// </returns>
439 [MethodImpl(MethodImplOptions.AggressiveInlining)]
440 public 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/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<TLinkAddr
42             ↪ ess> constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
43             ↪ base(constants, linksDataParts, linksIndexParts, header) { }
44
45         /// <summary>
46         /// <para>
47         /// Gets the left reference using the specified node.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="node">
52         /// <para>The node.</para>
53         /// <para></para>
54         /// </param>
55         /// <returns>
56         /// <para>The ref link</para>
57         /// <para></para>
58         /// </returns>
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)]

```



```

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     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLinkAddress GetLeft(TLinkAddress node) =>
92         ↪ GetLinkIndexPartReference(node).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        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(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        ↪ GetLinkIndexPartReference(node).RightAsSource = right;

```

```

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

```

```

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
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(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
        ↪ GreaterThan(firstTarget, secondTarget));

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

```

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

```

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

```

```

6 {
7     /// <summary>
8     /// <para>
9     /// Represents the external links sources size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLinkAddress}"/>
14    public unsafe class ExternalLinksSourcesSizeBalancedTreeMethods<TLinkAddress> :
15        ↳ ExternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="ExternalLinksSourcesSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="linksDataParts">
28        /// <para>A links data parts.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="linksIndexParts">
32        /// <para>A links index parts.</para>
33        /// <para></para>
34        /// </param>
35        /// <param name="header">
36        /// <para>A header.</para>
37        /// <para></para>
38        /// </param>
39        [MethodImpl(MethodImplOptions.AggressiveInlining)]
40        public ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
41            ↳ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
42            ↳ base(constants, linksDataParts, linksIndexParts, header) { }
43
44        /// <summary>
45        /// <para>
46        /// Gets the left reference using the specified node.
47        /// </para>
48        /// <para></para>
49        /// </summary>
50        /// <param name="node">
51        /// <para>The node.</para>
52        /// <para></para>
53        /// </param>
54        /// <returns>
55        /// <para>The ref link</para>
56        /// <para></para>
57        /// </returns>
58        [MethodImpl(MethodImplOptions.AggressiveInlining)]
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>
81        /// <para>
82        /// Gets the left using the specified node.
83        /// </para>

```

```

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

```

```

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

```

```

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

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

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

```

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

```

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

```

```

14 public unsafe class ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
    ↳ 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<TLinkAddr
    ↳ ess> constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
    ↳ base(constants, linksDataParts, linksIndexParts, header) { }
40
41     /// <summary>
42     /// <para>
43     /// Gets the left reference using the specified node.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="node">
48     /// <para>The node.</para>
49     /// <para></para>
50     /// </param>
51     /// <returns>
52     /// <para>The ref link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
    ↳ GetLinkIndexPartReference(node).LeftAsTarget;
57
58     /// <summary>
59     /// <para>
60     /// Gets the right reference using the specified node.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="node">
65     /// <para>The node.</para>
66     /// <para></para>
67     /// </param>
68     /// <returns>
69     /// <para>The ref link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↳ GetLinkIndexPartReference(node).RightAsTarget;
74
75     /// <summary>
76     /// <para>
77     /// Gets the left using the specified node.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <param name="node">
82     /// <para>The node.</para>
83     /// <para></para>
84     /// </param>

```



```

85     /// <returns>
86     /// <para>The link</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     protected override TLinkAddress GetLeft(TLinkAddress node) =>
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>
161    [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.
179     /// </para>
180     /// </summary>
181     /// <returns>
182     /// <para>The link</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     /// </summary>
192     /// <param name="link">
193     /// <para>The link.</para>
194     /// </param>
195     /// <returns>
196     /// <para>The link</para>
197     /// </returns>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
200         ↪ GetLinkDataPartReference(link).Target;
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance first is to the left of second.
205     /// </para>
206     /// </summary>
207     /// <param name="firstSource">
208     /// <para>The first source.</para>
209     /// </param>
210     /// <param name="firstTarget">
211     /// <para>The first target.</para>
212     /// </param>
213     /// <param name="secondSource">
214     /// <para>The second source.</para>
215     /// </param>
216     /// <param name="secondTarget">
217     /// <para>The second target.</para>
218     /// </param>
219     /// <returns>
220     /// <para>The bool</para>
221     /// </returns>

```

```

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

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

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

```

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

```

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

```

```

18     /// Initializes a new <see cref="ExternalLinksTargetsSizeBalancedTreeMethods"/> instance.
19     /// </para>
20     /// </summary>
21     /// <param name="constants">
22     /// <para>A constants.</para>
23     /// </param>
24     /// <param name="linksDataParts">
25     /// <para>A links data parts.</para>
26     /// </param>
27     /// <param name="linksIndexParts">
28     /// <para>A links index parts.</para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// </param>
33     [MethodImpl(MethodImplOptions.AggressiveInlining)]
34     public ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
35     ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
36     ↪ base(constants, linksDataParts, linksIndexParts, 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     /// </param>
46     /// <returns>
47     /// <para>The ref link</para>
48     /// </returns>
49     [MethodImpl(MethodImplOptions.AggressiveInlining)]
50     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
51     ↪ GetLinkIndexPartReference(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     /// </param>
61     /// <returns>
62     /// <para>The ref link</para>
63     /// </returns>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
66     ↪ GetLinkIndexPartReference(node).RightAsTarget;
67
68     /// <summary>
69     /// <para>
70     /// Gets the left using the specified node.
71     /// </para>
72     /// </summary>
73     /// <param name="node">
74     /// <para>The node.</para>
75     /// </param>
76     /// <returns>
77     /// <para>The link</para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     protected override TLinkAddress GetLeft(TLinkAddress node) =>
81     ↪ GetLinkIndexPartReference(node).LeftAsTarget;

```

```

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

```

```

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)]
236     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
        ↪ LessThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
        ↪ LessThan(firstSource, secondSource));

```

```

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

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

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.Split.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the internal links recursionless size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLinkAddress}"/>
21     /// <seealso cref="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(LinkConstants<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>
172    /// <param name="link">
173    /// <para>The link.</para>
174    /// <para></para>
175    /// </param>
176    /// <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>
    ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
    ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
    ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));
178
179 /// <summary>
180 /// <para>
181 /// Determines whether this instance first is to the left of second.
182 /// </para>
183 /// <para></para>
184 /// </summary>
185 /// <param name="first">
186 /// <para>The first.</para>
187 /// <para></para>
188 /// </param>
189 /// <param name="second">
190 /// <para>The second.</para>
191 /// <para></para>
192 /// </param>
193 /// <returns>
194 /// <para>The bool</para>
195 /// <para></para>
196 /// </returns>
197 [MethodImpl(MethodImplOptions.AggressiveInlining)]
198 protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress
    ↪ second) => LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
199
200 /// <summary>
201 /// <para>
202 /// Determines whether this instance first is to the right of second.
203 /// </para>
204 /// <para></para>
205 /// </summary>
206 /// <param name="first">
207 /// <para>The first.</para>
208 /// <para></para>
209 /// </param>
210 /// <param name="second">
211 /// <para>The second.</para>
212 /// <para></para>
213 /// </param>
214 /// <returns>
215 /// <para>The bool</para>
216 /// <para></para>
217 /// </returns>
218 [MethodImpl(MethodImplOptions.AggressiveInlining)]
219 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
    ↪ second) => GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
220
221 /// <summary>
222 /// <para>
223 /// Gets the link values using the specified link index.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="linkIndex">
228 /// <para>The link index.</para>
229 /// <para></para>
230 /// </param>
231 /// <returns>
232 /// <para>A list of t link</para>
233 /// <para></para>
234 /// </returns>
235 [MethodImpl(MethodImplOptions.AggressiveInlining)]
236 protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
237 {
238     ref var link = ref GetLinkDataPartReference(linkIndex);
239     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
240 }
241
242 /// <summary>
243 /// <para>
244 /// The zero.

```

```

245 /// </para>
246 /// <para></para>
247 /// </summary>
248 public TLinkAddress this[TLinkAddress link, TLinkAddress index]
249 {
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     get
252     {
253         var root = GetTreeRoot(link);
254         if (GreaterOrEqualThan(index, GetSize(root)))
255         {
256             return Zero;
257         }
258         while (!EqualToZero(root))
259         {
260             var left = GetLeftOrDefault(root);
261             var leftSize = GetSizeOrZero(left);
262             if (LessThan(index, leftSize))
263             {
264                 root = left;
265                 continue;
266             }
267             if (AreEqual(index, leftSize))
268             {
269                 return root;
270             }
271             root = GetRightOrDefault(root);
272             index = Subtract(index, Increment(leftSize));
273         }
274         return Zero; // TODO: Impossible situation exception (only if tree structure
275                     ↪ broken)
276     }
277 }
278 /// <summary>
279 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
280 /// ↪ (концом).
281 /// </summary>
282 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
283 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
284 /// <returns>Индекс искомой связи.</returns>
285 [MethodImpl(MethodImplOptions.AggressiveInlining)]
286 public abstract TLinkAddress Search(TLinkAddress source, TLinkAddress target);
287
288 /// <summary>
289 /// <para>
290 /// Searches the core using the specified root.
291 /// </para>
292 /// <para></para>
293 /// </summary>
294 /// <param name="root">
295 /// <para>The root.</para>
296 /// <para></para>
297 /// </param>
298 /// <param name="key">
299 /// <para>The key.</para>
300 /// <para></para>
301 /// </param>
302 /// <returns>
303 /// <para>The zero.</para>
304 /// <para></para>
305 /// </returns>
306 [MethodImpl(MethodImplOptions.AggressiveInlining)]
307 protected TLinkAddress SearchCore(TLinkAddress root, TLinkAddress key)
308 {
309     while (!EqualToZero(root))
310     {
311         var rootKey = GetKeyPartValue(root);
312         if (LessThan(key, rootKey)) // node.Key < root.Key
313         {
314             root = GetLeftOrDefault(root);
315         }
316         else if (GreaterThan(key, rootKey)) // node.Key > root.Key
317         {
318             root = GetRightOrDefault(root);
319         }
320         else // node.Key == root.Key

```

```

321         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>

```

```

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

```

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

```

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

```

```

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);
128
129    /// <summary>
130    /// <para>
131    /// Gets the key part value using the specified link.
132    /// </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>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress
207     ↪ second) => LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
208
209     /// <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) =>
        ↳ EachUsageCore(@base, GetTreeRoot(@base), handler);
365
366     // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
        ↳ low-level MSIL stack.
367     [MethodImpl(MethodImplOptions.AggressiveInlining)]
368     private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
        ↳ ReadHandler<TLinkAddress>? handler)
369     {
370         var @continue = Continue;
371         if (EqualToZero(link))
372         {
373             return @continue;
374         }
375         var @break = Break;
376         if (AreEqual(EachUsageCore(@base, GetLeftOrElseDefault(link), handler), @break))
377         {
378             return @break;
379         }
380         if (AreEqual(handler(GetLinkValues(link)), @break))
381         {
382             return @break;
383         }
384         if (AreEqual(EachUsageCore(@base, GetRightOrElseDefault(link), handler), @break))
385         {
386             return @break;
387         }
388         return @continue;
389     }
390
391     /// <summary>
392     /// <para>
393     /// Prints the node value using the specified node.
394     /// </para>
395     /// <para></para>
396     /// </summary>
397     /// <param name="node">
398     /// <para>The node.</para>
399     /// <para></para>
400     /// </param>
401     /// <param name="sb">
402     /// <para>The sb.</para>
403     /// <para></para>
404     /// </param>
405     [MethodImpl(MethodImplOptions.AggressiveInlining)]
406     protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
407     {
408         ref var link = ref GetLinkDataPartReference(node);
409         sb.Append(' ');
410         sb.Append(link.Source);
411         sb.Append('-');
412         sb.Append('>');
413         sb.Append(link.Target);
414     }
415 }
416 }

```

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

```

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

```

```

12 {
13     /// <summary>
14     /// <para>
15     /// Represents the internal links sources linked list methods.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="RelativeCircularDoublyLinkedListMethods{TLinkAddress}"/>
20     public unsafe class InternalLinksSourcesLinkedListMethods<TLinkAddress> :
        ↳ RelativeCircularDoublyLinkedListMethods<TLinkAddress>
21     {
22         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
        ↳ = UncheckedConverter<TLinkAddress, long>.Default;
23         private readonly byte* _linksDataParts;
24         private readonly byte* _linksIndexParts;
25         /// <summary>
26         /// <para>
27         /// The break.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         protected readonly TLinkAddress Break;
32         /// <summary>
33         /// <para>
34         /// The continue.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         protected readonly TLinkAddress Continue;
39
40         /// <summary>
41         /// <para>
42         /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="constants">
47         /// <para>A constants.</para>
48         /// <para></para>
49         /// </param>
50         /// <param name="linksDataParts">
51         /// <para>A links data parts.</para>
52         /// <para></para>
53         /// </param>
54         /// <param name="linksIndexParts">
55         /// <para>A links index parts.</para>
56         /// <para></para>
57         /// </param>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress> constants,
        ↳ byte* linksDataParts, byte* linksIndexParts)
60         {
61             _linksDataParts = linksDataParts;
62             _linksIndexParts = linksIndexParts;
63             Break = constants.Break;
64             Continue = constants.Continue;
65         }
66
67         /// <summary>
68         /// <para>
69         /// Gets the link data part reference using the specified link.
70         /// </para>
71         /// <para></para>
72         /// </summary>
73         /// <param name="link">
74         /// <para>The link.</para>
75         /// <para></para>
76         /// </param>
77         /// <returns>
78         /// <para>A ref raw link data part of t link</para>
79         /// <para></para>
80         /// </returns>
81         [MethodImpl(MethodImplOptions.AggressiveInlining)]
82         protected virtual ref RawLinkDataPart<TLinkAddress>
        ↳ GetLinkDataPartReference(TLinkAddress link) => ref
        ↳ AsRef<RawLinkDataPart<TLinkAddress>>(_linksDataParts +
        ↳ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
        ↳ _addressToInt64Converter.Convert(link)));

```

```

84     /// <summary>
85     /// <para>
86     /// Gets the link index part reference using the specified link.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="link">
91     /// <para>The link.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>A ref raw link index part of t link</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected virtual ref RawLinkIndexPart<TLinkAddress>
100     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
101     ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(_linksIndexParts +
102     ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
103     ↪ _addressToInt64Converter.Convert(link)));
104
105     /// <summary>
106     /// <para>
107     /// Gets the first using the specified head.
108     /// </para>
109     /// <para></para>
110     /// </summary>
111     /// <param name="head">
112     /// <para>The head.</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 GetFirst(TLinkAddress head) =>
121     ↪ GetLinkIndexPartReference(head).RootAsSource;
122
123     /// <summary>
124     /// <para>
125     /// Gets the last using the specified head.
126     /// </para>
127     /// <para></para>
128     /// </summary>
129     /// <param name="head">
130     /// <para>The head.</para>
131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The link</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected override TLinkAddress GetLast(TLinkAddress head)
139     {
140         var first = GetLinkIndexPartReference(head).RootAsSource;
141         if (EqualToZero(first))
142         {
143             return first;
144         }
145         else
146         {
147             return GetPrevious(first);
148         }
149     }
150
151     /// <summary>
152     /// <para>
153     /// Gets the previous using the specified element.
154     /// </para>
155     /// <para></para>
156     /// </summary>
157     /// <param name="element">
158     /// <para>The element.</para>
159     /// <para></para>
160     /// </param>
161     /// <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     /// </summary>
169     /// <param name="element">
170     /// <para>The element.</para>
171     /// </param>
172     /// </returns>
173     /// <para>The link</para>
174     /// <para></para>
175     /// </returns>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override TLinkAddress GetNext(TLinkAddress element) =>
178         ↪ GetLinkIndexPartReference(element).RightAsSource;
179
180     /// <summary>
181     /// <para>
182     /// Gets the size using the specified head.
183     /// </para>
184     /// </summary>
185     /// <param name="head">
186     /// <para>The head.</para>
187     /// </param>
188     /// </returns>
189     /// <para>The link</para>
190     /// <para></para>
191     /// </returns>
192     [MethodImpl(MethodImplOptions.AggressiveInlining)]
193     protected override TLinkAddress GetSize(TLinkAddress head) =>
194         ↪ GetLinkIndexPartReference(head).SizeAsSource;
195
196     /// <summary>
197     /// <para>
198     /// Sets the first using the specified head.
199     /// </para>
200     /// </summary>
201     /// <param name="head">
202     /// <para>The head.</para>
203     /// </param>
204     /// <param name="element">
205     /// <para>The element.</para>
206     /// </param>
207     /// </returns>
208     [MethodImpl(MethodImplOptions.AggressiveInlining)]
209     protected override void SetFirst(TLinkAddress head, TLinkAddress element) =>
210         ↪ GetLinkIndexPartReference(head).RootAsSource = element;
211
212     /// <summary>
213     /// <para>
214     /// Sets the last using the specified head.
215     /// </para>
216     /// </summary>
217     /// <param name="head">
218     /// <para>The head.</para>
219     /// </param>
220     /// <param name="element">
221     /// <para>The element.</para>
222     /// </param>
223     /// </returns>
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     protected override void SetLast(TLinkAddress head, TLinkAddress element)
226     {
227

```

```

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">
303     /// <para>The head.</para>
304     /// <para></para>
305     /// </param>
306     /// <returns>
307     /// <para>The link</para>
308     /// <para></para>

```

```

306     /// </returns>
307     [MethodImpl(MethodImplOptions.AggressiveInlining)]
308     public TLinkAddress CountUsages(TLinkAddress head) => GetSize(head);
309
310     /// <summary>
311     /// <para>
312     /// Gets the link values using the specified link index.
313     /// </para>
314     /// <para></para>
315     /// </summary>
316     /// <param name="linkIndex">
317     /// <para>The link index.</para>
318     /// <para></para>
319     /// </param>
320     /// <returns>
321     /// <para>A list of t link</para>
322     /// <para></para>
323     /// </returns>
324     [MethodImpl(MethodImplOptions.AggressiveInlining)]
325     protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
326     {
327         ref var link = ref GetLinkDataPartReference(linkIndex);
328         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
329     }
330
331     /// <summary>
332     /// <para>
333     /// Eaches the usage using the specified source.
334     /// </para>
335     /// <para></para>
336     /// </summary>
337     /// <param name="source">
338     /// <para>The source.</para>
339     /// <para></para>
340     /// </param>
341     /// <param name="handler">
342     /// <para>The handler.</para>
343     /// <para></para>
344     /// </param>
345     /// <returns>
346     /// <para>The continue.</para>
347     /// <para></para>
348     /// </returns>
349     [MethodImpl(MethodImplOptions.AggressiveInlining)]
350     public TLinkAddress EachUsage(TLinkAddress source, ReadHandler<TLinkAddress>? handler)
351     {
352         var @continue = Continue;
353         var @break = Break;
354         var current = GetFirst(source);
355         var first = current;
356         while (!EqualToZero(current))
357         {
358             if (AreEqual(handler(GetLinkValues(current)), @break))
359             {
360                 return @break;
361             }
362             current = GetNext(current);
363             if (AreEqual(current, first))
364             {
365                 return @continue;
366             }
367         }
368         return @continue;
369     }
370 }
371 }

```

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

```

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

```

```

12  /// </summary>
13  /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
14  public unsafe class InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
15  {
16      /// <summary>
17      /// <para>
18      /// Initializes a new <see
    ↪ cref="InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddr
    ↪ ess> constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
    ↪ base(constants, linksDataParts, linksIndexParts, header) { }
40
41      /// <summary>
42      /// <para>
43      /// Gets the left reference using the specified node.
44      /// </para>
45      /// <para></para>
46      /// </summary>
47      /// <param name="node">
48      /// <para>The node.</para>
49      /// <para></para>
50      /// </param>
51      /// <returns>
52      /// <para>The ref link</para>
53      /// <para></para>
54      /// </returns>
55      [MethodImpl(MethodImplOptions.AggressiveInlining)]
56      protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
    ↪ GetLinkIndexPartReference(node).LeftAsSource;
57
58      /// <summary>
59      /// <para>
60      /// Gets the right reference using the specified node.
61      /// </para>
62      /// <para></para>
63      /// </summary>
64      /// <param name="node">
65      /// <para>The node.</para>
66      /// <para></para>
67      /// </param>
68      /// <returns>
69      /// <para>The ref link</para>
70      /// <para></para>
71      /// </returns>
72      [MethodImpl(MethodImplOptions.AggressiveInlining)]
73      protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↪ GetLinkIndexPartReference(node).RightAsSource;
74
75      /// <summary>
76      /// <para>
77      /// Gets the left using the specified node.
78      /// </para>
79      /// <para></para>
80      /// </summary>
81      /// <param name="node">
82      /// <para>The node.</para>

```



```

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

```

```

157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 protected override TLinkAddress GetSize(TLinkAddress node) =>
    ↳ 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) =>
    ↳ 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) =>
    ↳ GetLinkIndexPartReference(link).RootAsSource;

193
194 /// <summary>
195 /// <para>
196 /// Gets the base part value 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>The link</para>
206 /// <para></para>
207 /// </returns>
208 [MethodImpl(MethodImplOptions.AggressiveInlining)]
209 protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
    ↳ GetLinkDataPartReference(link).Source;

210
211 /// <summary>
212 /// <para>
213 /// Gets the key part value using the specified link.
214 /// </para>
215 /// <para></para>
216 /// </summary>
217 /// <param name="link">
218 /// <para>The link.</para>
219 /// <para></para>
220 /// </param>
221 /// <returns>
222 /// <para>The link</para>
223 /// <para></para>
224 /// </returns>
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 protected override 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) =>
266         ↪ SearchCore(GetTreeRoot(source), target);
267 }

```

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

```

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

```

```

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) =>
    ↪ GetLinkIndexPartReference(node).RightAsSource;

```

```

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

```

```

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>
254     /// <para></para>
255     /// </summary>
256     /// <param name="source">
257     /// <para>The source.</para>
258     /// <para></para>
259     /// </param>

```

```

257     /// <param name="target">
258     /// <para>The target.</para>
259     /// <para></para>
260     /// </param>
261     /// <returns>
262     /// <para>The link</para>
263     /// <para></para>
264     /// </returns>
265     public override TLinkAddress Search(TLinkAddress source, TLinkAddress target) =>
        ↳ SearchCore(GetTreeRoot(source), target);
266 }
267 }

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.Split.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the internal links targets recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
14    public unsafe class InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
        ↳ 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<TLinkAddress> constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
41        ↳ base(constants, linksDataParts, linksIndexParts, header) { }
42
43        /// <summary>
44        /// <para>
45        /// Gets the left reference using the specified node.
46        /// </para>
47        /// <para></para>
48        /// </summary>
49        /// <param name="node">
50        /// <para>The node.</para>
51        /// <para></para>
52        /// </param>
53        /// <returns>
54        /// <para>The ref link</para>
55        /// <para></para>
56        /// </returns>
57        [MethodImpl(MethodImplOptions.AggressiveInlining)]
58        protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
59        ↳ GetLinkIndexPartReference(node).LeftAsTarget;
60
61        /// <summary>
62        /// <para>

```

```

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

```



```

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

```

```

208 [MethodImpl(MethodImplOptions.AggressiveInlining)]
209 protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
    ↳ GetLinkDataPartReference(link).Target;
210
211 /// <summary>
212 /// <para>
213 /// Gets the key part value using the specified link.
214 /// </para>
215 /// <para></para>
216 /// </summary>
217 /// <param name="link">
218 /// <para>The link.</para>
219 /// <para></para>
220 /// </param>
221 /// <returns>
222 /// <para>The link</para>
223 /// <para></para>
224 /// </returns>
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 protected override TLinkAddress GetKeyPartValue(TLinkAddress link) =>
    ↳ 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) =>
    ↳ SearchCore(GetTreeRoot(target), source);
266 }
267 }

```

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

```

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

```

```

14 public unsafe class InternalLinksTargetsSizeBalancedTreeMethods<TLinkAddress> :
    ↳ InternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>
15 {
16     /// <summary>
17     /// <para>
18     /// Initializes a new <see cref="InternalLinksTargetsSizeBalancedTreeMethods"/> instance.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="linksDataParts">
27     /// <para>A links data parts.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="linksIndexParts">
31     /// <para>A links index parts.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="header">
35     /// <para>A header.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
        ↳ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
        ↳ base(constants, linksDataParts, linksIndexParts, header) { }
40
41     /// <summary>
42     /// <para>
43     /// Gets the left reference using the specified node.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="node">
48     /// <para>The node.</para>
49     /// <para></para>
50     /// </param>
51     /// <returns>
52     /// <para>The ref link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
        ↳ 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>
161    [MethodImpl(MethodImplOptions.AggressiveInlining)]
162    protected override TLinkAddress GetSize(TLinkAddress node) =>
163        ↪ 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 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).RootAsTarget;
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).Target;
213
214    /// <summary>
215    /// <para>
216    /// Gets the key part value using the specified link.
217    /// </para>
218    /// <para></para>
219    /// </summary>
220    /// <param name="link">
221    /// <para>The link.</para>
222    /// <para></para>
223    /// </param>
224    /// <returns>
225    /// <para>The link</para>
226    /// <para></para>
227    /// </returns>
228    [MethodImpl(MethodImplOptions.AggressiveInlining)]
229    protected override TLinkAddress GetKeyPartValue(TLinkAddress link) =>
230        ↪ GetLinkDataPartReference(link).Source;
231
232    /// <summary>
233    /// <para>
234    /// Clears the node using the specified node.
235    /// </para>
236    /// <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) =>
        ↪ SearchCore(GetTreeRoot(target), source);
266
267 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using static System.Runtime.CompilerServices.Unsafe;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.Split.Generic
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the split memory links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="SplitMemoryLinksBase{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 /// <param name="dataMemory">
114 /// <para>A data memory.</para>
115 /// </param>
116 /// <param name="indexMemory">
117 /// <para>A index memory.</para>
118 /// </param>
119 /// <param name="memoryReservationStep">
120 /// <para>A memory reservation step.</para>
121 /// </param>
122 /// <param name="constants">
123 /// <para>A constants.</para>
124 /// </param>
125 /// <param name="indexTreeType">
126 /// <para>A index tree type.</para>
127 /// </param>
128 /// <param name="useLinkedList">
129 /// <para>A use linked list.</para>
130 /// </param>
131 [MethodImpl(MethodImplOptions.AggressiveInlining)]
132 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
133     ↪ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants,
134     ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
135     ↪ memoryReservationStep, constants, useLinkedList)
136 {
137     if (indexTreeType == IndexTreeType.SizeBalancedTree)
138     {
139         _createInternalSourceTreeMethods = () => new
140             ↪ InternalLinksSourcesSizeBalancedTreeMethods<TLinkAddress>(Constants,
141             ↪ _linksDataParts, _linksIndexParts, _header);
142         _createExternalSourceTreeMethods = () => new
143             ↪ ExternalLinksSourcesSizeBalancedTreeMethods<TLinkAddress>(Constants,
144             ↪ _linksDataParts, _linksIndexParts, _header);
145         _createInternalTargetTreeMethods = () => new
146             ↪ InternalLinksTargetsSizeBalancedTreeMethods<TLinkAddress>(Constants,
147             ↪ _linksDataParts, _linksIndexParts, _header);
148         _createExternalTargetTreeMethods = () => new
149             ↪ ExternalLinksTargetsSizeBalancedTreeMethods<TLinkAddress>(Constants,
150             ↪ _linksDataParts, _linksIndexParts, _header);
151     }
152     else
153     {
154         _createInternalSourceTreeMethods = () => new InternalLinksSourcesRecursionlessSi
155             ↪ zeBalancedTreeMethods<TLinkAddress>(Constants, _linksDataParts,
156             ↪ _linksIndexParts, _header);
157         _createExternalSourceTreeMethods = () => new ExternalLinksSourcesRecursionlessSi
158             ↪ zeBalancedTreeMethods<TLinkAddress>(Constants, _linksDataParts,
159             ↪ _linksIndexParts, _header);
160         _createInternalTargetTreeMethods = () => new InternalLinksTargetsRecursionlessSi
161             ↪ zeBalancedTreeMethods<TLinkAddress>(Constants, _linksDataParts,
162             ↪ _linksIndexParts, _header);
163         _createExternalTargetTreeMethods = () => new ExternalLinksTargetsRecursionlessSi
164             ↪ zeBalancedTreeMethods<TLinkAddress>(Constants, _linksDataParts,
165             ↪ _linksIndexParts, _header);
166     }
167     Init(dataMemory, indexMemory);
168 }
169
170 /// <summary>
171 /// <para>
172 /// Sets the pointers using the specified data memory.
173 /// </para>
174 /// </summary>
175 /// <param name="dataMemory">
176 /// <para>The data memory.</para>
177 /// </param>

```



```

168 /// <param name="indexMemory">
169 /// <para>The index memory.</para>
170 /// </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 /// </summary>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override void ResetPointers()
203 {
204     base.ResetPointers();
205     _linksDataParts = null;
206     _linksIndexParts = null;
207     _header = null;
208 }
209
210 /// <summary>
211 /// <para>
212 /// Gets the header reference.
213 /// </para>
214 /// </summary>
215 /// <para></para>
216 /// </summary>
217 /// <returns>
218 /// <para>A ref links header of t link</para>
219 /// </returns>
220 [MethodImpl(MethodImplOptions.AggressiveInlining)]
221 protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
222 ↪ AsRef<LinksHeader<TLinkAddress>>(_header);
223
224 /// <summary>
225 /// <para>
226 /// Gets the link data part reference using the specified link index.
227 /// </para>
228 /// </summary>
229 /// <para></para>
230 /// </summary>
231 /// <param name="linkIndex">
232 /// <para>The link index.</para>
233 /// </para>
234 /// </param>
235 /// <returns>
236 /// <para>A ref raw link data part of t link</para>
237 /// </returns>
238 [MethodImpl(MethodImplOptions.AggressiveInlining)]
239 protected override ref RawLinkDataPart<TLinkAddress>
240 ↪ GetLinkDataPartReference(TLinkAddress linkIndex) => ref
241 ↪ AsRef<RawLinkDataPart<TLinkAddress>>(_linksDataParts + (LinkDataPartSizeInBytes *
242 ↪ ConvertToInt64(linkIndex)));
243
244 /// <summary>

```

```

238     /// <para>
239     /// Gets the link index part reference using the specified link index.
240     /// </para>
241     /// <para></para>
242     /// </summary>
243     /// <param name="linkIndex">
244     /// <para>The link index.</para>
245     /// <para></para>
246     /// </param>
247     /// <returns>
248     /// <para>A ref raw link index part of t link</para>
249     /// <para></para>
250     /// </returns>
251     [MethodImpl(MethodImplOptions.AggressiveInlining)]
252     protected override ref RawLinkIndexPart<TLinkAddress>
        ↪ GetLinkIndexPartReference(TLinkAddress linkIndex) => ref
        ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(_linksIndexParts + (LinkIndexPartSizeInBytes *
        ↪ ConvertToInt64(linkIndex)));
253 }
254 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Disposables;
5  using Platform.Singletons;
6  using Platform.Converters;
7  using Platform.Numbers;
8  using Platform.Memory;
9  using Platform.Data.Exceptions;
10 using Platform.Delegates;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Memory.Split.Generic
15 {
16     /// <summary>
17     /// <para>
18     /// Represents the split memory links base.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <seealso cref="DisposableBase"/>
23     /// <seealso cref="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 =
55         ↳ LinksHeader<TLinkAddress>.SizeInBytes;
56
57     /// <summary>
58     /// <para>
59     /// The default links size step.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
64
65     /// <summary>
66     /// <para>
67     /// The data memory.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     protected readonly IResizableDirectMemory _dataMemory;
72     /// <summary>
73     /// <para>
74     /// The index memory.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     protected readonly IResizableDirectMemory _indexMemory;
79     /// <summary>
80     /// <para>
81     /// The use linked list.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     protected readonly bool _useLinkedList;
86     /// <summary>
87     /// <para>
88     /// The data memory reservation step in bytes.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     protected readonly long _dataMemoryReservationStepInBytes;
93     /// <summary>
94     /// <para>
95     /// The index memory reservation step in bytes.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     protected readonly long _indexMemoryReservationStepInBytes;
100
101     /// <summary>
102     /// <para>
103     /// The internal sources list methods.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     protected InternalLinksSourcesLinkedListMethods<TLinkAddress> InternalSourcesListMethods;
108     /// <summary>
109     /// <para>
110     /// The internal sources tree methods.
111     /// </para>
112     /// <para></para>
113     /// </summary>
114     protected ILinksTreeMethods<TLinkAddress> InternalSourcesTreeMethods;
115     /// <summary>
116     /// <para>
117     /// The external sources tree methods.
118     /// </para>
119     /// <para></para>
120     /// </summary>
121     protected ILinksTreeMethods<TLinkAddress> ExternalSourcesTreeMethods;
122     /// <summary>
123     /// <para>
124     /// The internal targets tree methods.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     protected ILinksTreeMethods<TLinkAddress> InternalTargetsTreeMethods;
129     /// <summary>
130     /// <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)
198     {
199         _dataMemory = dataMemory;
200         _indexMemory = indexMemory;
201         _dataMemoryReservationStepInBytes = memoryReservationStep * LinkDataPartSizeInBytes;
202         _indexMemoryReservationStepInBytes = memoryReservationStep *
203         // → LinkIndexPartSizeInBytes;
204         _useLinkedList = useLinkedList;
205         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<LinkConstants<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 = Convert.ToInt64(header.AllocatedLinks);
252     // Adjust reserved capacity
253     var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
254     if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
255     {
256         minimumDataReservedCapacity = dataMemory.UsedCapacity;
257     }
258     if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
259     {
260         minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
261     }
262     var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
263     if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
264     {
265         minimumIndexReservedCapacity = indexMemory.UsedCapacity;
266     }
267     if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
268     {
269         minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
270     }
271     // Check for alignment
272     if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
273     {
274         minimumDataReservedCapacity = ((minimumDataReservedCapacity /
    ↪ _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
    ↪ _dataMemoryReservationStepInBytes;
275     }

```

```

276     if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)
277     {
278         minimumIndexReservedCapacity = ((minimumIndexReservedCapacity /
279             ↪ _indexMemoryReservationStepInBytes) * _indexMemoryReservationStepInBytes) +
280             ↪ _indexMemoryReservationStepInBytes;
281     }
282     if (dataMemory.ReservedCapacity != minimumDataReservedCapacity)
283     {
284         dataMemory.ReservedCapacity = minimumDataReservedCapacity;
285     }
286     if (indexMemory.ReservedCapacity != minimumIndexReservedCapacity)
287     {
288         indexMemory.ReservedCapacity = minimumIndexReservedCapacity;
289     }
290     SetPointers(dataMemory, indexMemory);
291     header = ref GetHeaderReference();
292     // Ensure correctness _memory.UsedCapacity over _header->AllocatedLinks
293     // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
294     dataMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
295         ↪ LinkDataPartSizeInBytes) + LinkDataPartSizeInBytes; // First link is read only
296     ↪ zero link.
297     indexMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
298         ↪ LinkIndexPartSizeInBytes) + LinkHeaderSizeInBytes;
299     // Ensure correctness _memory.ReservedLinks over _header->ReservedCapacity
300     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
301     header.ReservedLinks = ConvertToAddress((dataMemory.ReservedCapacity -
302         ↪ LinkDataPartSizeInBytes) / LinkDataPartSizeInBytes);
303 }
304
305 /// <summary>
306 /// <para>
307 /// Counts the substitution.
308 /// </para>
309 /// <para></para>
310 /// </summary>
311 /// <param name="restriction">
312 /// <para>The substitution.</para>
313 /// <para></para>
314 /// </param>
315 /// <exception cref="NotSupportedException">
316 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
317 /// <para></para>
318 /// </exception>
319 /// <returns>
320 /// <para>The link</para>
321 /// <para></para>
322 /// </returns>
323 [MethodImpl(MethodImplOptions.AggressiveInlining)]
324 public virtual TLinkAddress Count(IList<TLinkAddress>? restriction)
325 {
326     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
327     if (restriction.Count == 0)
328     {
329         return Total;
330     }
331     var constants = Constants;
332     var any = constants.Any;
333     var index = this.GetIndex(restriction);
334     if (restriction.Count == 1)
335     {
336         if (AreEqual(index, any))
337         {
338             return Total;
339         }
340         return Exists(index) ? GetOne() : GetZero();
341     }
342     if (restriction.Count == 2)
343     {
344         var value = restriction[1];
345         if (AreEqual(index, any))
346         {
347             if (AreEqual(value, any))
348             {
349                 return Total; // Any - как отсутствие ограничения
350             }
351             var externalReferencesRange = constants.ExternalReferencesRange;
352             if (externalReferencesRange.HasValue &&
353                 ↪ externalReferencesRange.Value.Contains(value))

```

```

347     {
348         return Add(ExternalSourcesTreeMethods.CountUsages(value),
349             ↪ ExternalTargetsTreeMethods.CountUsages(value));
350     }
351     else
352     {
353         if (_useLinkedList)
354         {
355             return Add(InternalSourcesListMethods.CountUsages(value),
356                 ↪ InternalTargetsTreeMethods.CountUsages(value));
357         }
358         else
359         {
360             return Add(InternalSourcesTreeMethods.CountUsages(value),
361                 ↪ InternalTargetsTreeMethods.CountUsages(value));
362         }
363     }
364 }
365 else
366 {
367     if (!Exists(index))
368     {
369         return GetZero();
370     }
371     if (AreEqual(value, any))
372     {
373         return GetOne();
374     }
375     ref var storedLinkValue = ref GetLinkDataPartReference(index);
376     if (AreEqual(storedLinkValue.Source, value) ||
377         ↪ AreEqual(storedLinkValue.Target, value))
378     {
379         return GetOne();
380     }
381     return GetZero();
382 }
383 }
384 if (restriction.Count == 3)
385 {
386     var externalReferencesRange = constants.ExternalReferencesRange;
387     var source = this.GetSource(restriction);
388     var target = this.GetTarget(restriction);
389     if (AreEqual(index, any))
390     {
391         if (AreEqual(source, any) && AreEqual(target, any))
392         {
393             return Total;
394         }
395         else if (AreEqual(source, any))
396         {
397             if (externalReferencesRange.HasValue &&
398                 ↪ externalReferencesRange.Value.Contains(target))
399             {
400                 return ExternalTargetsTreeMethods.CountUsages(target);
401             }
402             else
403             {
404                 return InternalTargetsTreeMethods.CountUsages(target);
405             }
406         }
407         else if (AreEqual(target, any))
408         {
409             if (externalReferencesRange.HasValue &&
410                 ↪ externalReferencesRange.Value.Contains(source))
411             {
412                 return ExternalSourcesTreeMethods.CountUsages(source);
413             }
414             else
415             {
416                 if (_useLinkedList)
417                 {
418                     return InternalSourcesListMethods.CountUsages(source);
419                 }
420                 else
421                 {
422                     return InternalSourcesTreeMethods.CountUsages(source);
423                 }
424             }
425         }
426     }
427 }

```

```

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     {
490         if (AreEqual(storedLinkValue.Source, source) &&
491             ↪ AreEqual(storedLinkValue.Target, target))
492         {
493             return GetOne();
494         }
495     }
496     return GetZero();
497 }
498 var value = default(TLinkAddress);

```



```

491         if (AreEqual(source, any))
492         {
493             value = target;
494         }
495         if (AreEqual(target, any))
496         {
497             value = source;
498         }
499         if (AreEqual(storedLinkValue.Source, value) ||
500             ↪ AreEqual(storedLinkValue.Target, value))
501         {
502             return GetOne();
503         }
504         return GetZero();
505     }
506     throw new NotSupportedException("Другие размеры и способы ограничений не
507     ↪ поддерживаются.");
508 }
509
510 /// <summary>
511 /// <para>
512 /// Eaches the handler.
513 /// </para>
514 /// <para></para>
515 /// </summary>
516 /// <param name="handler">
517 /// <para>The handler.</para>
518 /// </param>
519 /// <param name="restriction">
520 /// <para>The substitution.</para>
521 /// </param>
522 /// <exception cref="NotSupportedException">
523 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
524 /// </exception>
525 /// </returns>
526 /// <para>The link</para>
527 /// </returns>
528 [MethodImpl(MethodImplOptions.AggressiveInlining)]
529 public virtual TLinkAddress Each(IList<TLinkAddress>? restriction,
530     ↪ ReadHandler<TLinkAddress>? handler)
531 {
532     var constants = Constants;
533     var @break = constants.Break;
534     if (restriction.Count == 0)
535     {
536         for (var link = GetOne(); LessOrEqualThan(link,
537             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
538         {
539             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
540             {
541                 return @break;
542             }
543         }
544         return @break;
545     }
546     var @continue = constants.Continue;
547     var any = constants.Any;
548     var index = this.GetIndex(restriction);
549     if (restriction.Count == 1)
550     {
551         if (AreEqual(index, any))
552         {
553             return Each(Array.Empty<TLinkAddress>(), handler);
554         }
555         if (!Exists(index))
556         {
557             return @continue;
558         }
559         return handler(GetLinkStruct(index));
560     }
561     if (restriction.Count == 2)
562     {
563         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 if (restriction.Count == 3)
597 {
598     var externalReferencesRange = constants.ExternalReferencesRange;
599     var source = this.GetSource(restriction);
600     var target = this.GetTarget(restriction);
601     if (AreEqual(index, any))
602     {
603         if (AreEqual(source, any) && AreEqual(target, any))
604         {
605             return Each(Array.Empty<TLinkAddress>(), handler);
606         }
607         else if (AreEqual(source, any))
608         {
609             if (externalReferencesRange.HasValue &&
610                 ↪ externalReferencesRange.Value.Contains(target))
611             {
612                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
613             }
614             else
615             {
616                 return InternalTargetsTreeMethods.EachUsage(target, handler);
617             }
618         }
619         else if (AreEqual(target, any))
620         {
621             if (externalReferencesRange.HasValue &&
622                 ↪ externalReferencesRange.Value.Contains(source))
623             {
624                 return ExternalSourcesTreeMethods.EachUsage(source, handler);
625             }
626             else
627             {
628                 if (_useLinkedList)
629                 {
630                     return InternalSourcesListMethods.EachUsage(source, handler);
631                 }
632                 else
633                 {
634                     return InternalSourcesTreeMethods.EachUsage(source, handler);
635                 }
636             }
637         }
638     }
639     else //if(source != Any && target != Any)
640     {
641         TLinkAddress link;
642         if (externalReferencesRange.HasValue)

```

```

640 {
641     if (externalReferencesRange.Value.Contains(source) &&
        ↳ externalReferencesRange.Value.Contains(target))
642     {
643         link = ExternalSourcesTreeMethods.Search(source, target);
644     }
645     else if (externalReferencesRange.Value.Contains(source))
646     {
647         link = InternalTargetsTreeMethods.Search(source, target);
648     }
649     else if (externalReferencesRange.Value.Contains(target))
650     {
651         if (_useLinkedList)
652         {
653             link = ExternalSourcesTreeMethods.Search(source, target);
654         }
655         else
656         {
657             link = InternalSourcesTreeMethods.Search(source, target);
658         }
659     }
660     else
661     {
662         if (_useLinkedList ||
        ↳ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
        ↳ InternalTargetsTreeMethods.CountUsages(target)))
663         {
664             link = InternalTargetsTreeMethods.Search(source, target);
665         }
666         else
667         {
668             link = InternalSourcesTreeMethods.Search(source, target);
669         }
670     }
671 }
672 else
673 {
674     if (_useLinkedList ||
        ↳ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
        ↳ InternalTargetsTreeMethods.CountUsages(target)))
675     {
676         link = InternalTargetsTreeMethods.Search(source, target);
677     }
678     else
679     {
680         link = InternalSourcesTreeMethods.Search(source, target);
681     }
682 }
683 return AreEqual(link, constants.Null) ? @continue :
        ↳ handler(GetLinkStruct(link));
684 }
685 else
686 {
687     if (!Exists(index))
688     {
689         return @continue;
690     }
691     if (AreEqual(source, any) && AreEqual(target, any))
692     {
693         return handler(GetLinkStruct(index));
694     }
695     ref var storedLinkValue = ref GetLinkDataPartReference(index);
696     if (!AreEqual(source, any) && !AreEqual(target, any))
697     {
698         if (AreEqual(storedLinkValue.Source, source) &&
699             AreEqual(storedLinkValue.Target, target))
700         {
701             return handler(GetLinkStruct(index));
702         }
703         return @continue;
704     }
705     var value = default(TLinkAddress);
706     if (AreEqual(source, any))
707     {
708         value = target;
709     }
710     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 &&
            ↳ externalReferencesRange.Value.Contains(source))
778         {
779             ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
780         }

```

```

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

```

```

851 [MethodImpl(MethodImplOptions.AggressiveInlining)]
852 public virtual TLinkAddress Delete(IList<TLinkAddress>? restriction,
    ↳ WriteHandler<TLinkAddress>? handler)
853 {
854     ref var header = ref GetHeaderReference();
855     var link = restriction[Constants.IndexPart];
856     var before = GetLinkStruct(link);
857     if (LessThan(link, header.AllocatedLinks))
858     {
859         UnusedLinksListMethods.AttachAsFirst(link);
860     }
861     else if (AreEqual(link, header.AllocatedLinks))
862     {
863         header.AllocatedLinks = Decrement(header.AllocatedLinks);
864         _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
865         _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
866         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
            ↳ пока не дойдём до первой существующей связи
867         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
868         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
            ↳ IsUnusedLink(header.AllocatedLinks))
869         {
870             UnusedLinksListMethods.Detach(header.AllocatedLinks);
871             header.AllocatedLinks = Decrement(header.AllocatedLinks);
872             _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
873             _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
874         }
875     }
876     return handler != null ? handler(before, null) : Constants.Continue;
877 }
878
879 /// <summary>
880 /// <para>
881 /// Gets the link struct using the specified link index.
882 /// </para>
883 /// <para></para>
884 /// </summary>
885 /// <param name="linkIndex">
886 /// <para>The link index.</para>
887 /// <para></para>
888 /// </param>
889 /// <returns>
890 /// <para>A list of t link</para>
891 /// <para></para>
892 /// </returns>
893 [MethodImpl(MethodImplOptions.AggressiveInlining)]
894 public IList<TLinkAddress>? GetLinkStruct(TLinkAddress linkIndex)
895 {
896     ref var link = ref GetLinkDataPartReference(linkIndex);
897     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
898 }
899
900 /// <remarks>
901 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
    ↳ адрес реально поменялся
902 ///
903 /// Указатель this.links может быть в том же месте,
904 /// так как 0-я связь не используется и имеет такой же размер как Header,
905 /// поэтому header размещается в том же месте, что и 0-я связь
906 /// </remarks>
907 [MethodImpl(MethodImplOptions.AggressiveInlining)]
908 protected abstract void SetPointers(IResizableDirectMemory dataMemory,
    ↳ IResizableDirectMemory indexMemory);
909
910 /// <summary>
911 /// <para>
912 /// Resets the pointers.
913 /// </para>
914 /// <para></para>
915 /// </summary>
916 [MethodImpl(MethodImplOptions.AggressiveInlining)]
917 protected virtual void ResetPointers()
918 {
919     InternalSourcesListMethods = null;
920     InternalSourcesTreeMethods = null;
921     ExternalSourcesTreeMethods = null;
922     InternalTargetsTreeMethods = null;
923     ExternalTargetsTreeMethods = null;

```

```

924     UnusedLinksListMethods = null;
925 }
926
927 /// <summary>
928 /// <para>
929 /// Gets the header reference.
930 /// </para>
931 /// <para></para>
932 /// </summary>
933 /// <returns>
934 /// <para>A ref links header of t link</para>
935 /// <para></para>
936 /// </returns>
937 [MethodImpl(MethodImplOptions.AggressiveInlining)]
938 protected abstract ref LinksHeader<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)
994     && !IsUnusedLink(link);
995
996 /// <summary>
997 /// <para>
998 /// Determines whether this instance is unused link.
999 /// </para>
1000 /// <para></para>
1001 /// </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>
1070    /// </returns>
1071    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1072    protected virtual bool AreEqual(TLinkAddress first, TLinkAddress second) =>
1073        ↪ _equalityComparer.Equals(first, second);
1074
1075    /// <summary>
1076    /// <para>
1077    /// Determines whether this instance less than.

```



```

1074    /// </para>
1075    /// <para></para>
1076    /// </summary>
1077    /// <param name="first">
1078    /// <para>The first.</para>
1079    /// <para></para>
1080    /// </param>
1081    /// <param name="second">
1082    /// <para>The second.</para>
1083    /// <para></para>
1084    /// </param>
1085    /// <returns>
1086    /// <para>The bool</para>
1087    /// <para></para>
1088    /// </returns>
1089    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1090    protected virtual bool LessThan(TLinkAddress first, TLinkAddress second) =>
1091        ↪ _comparer.Compare(first, second) < 0;
1092
1093    /// <summary>
1094    /// <para>
1095    /// <para>Determines whether this instance less or equal than.
1096    /// </para>
1097    /// <para></para>
1098    /// </summary>
1099    /// <param name="first">
1100    /// <para>The first.</para>
1101    /// <para></para>
1102    /// </param>
1103    /// <param name="second">
1104    /// <para>The second.</para>
1105    /// <para></para>
1106    /// </param>
1107    /// <returns>
1108    /// <para>The bool</para>
1109    /// <para></para>
1110    /// </returns>
1111    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1112    protected virtual bool LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
1113        ↪ _comparer.Compare(first, second) <= 0;
1114
1115    /// <summary>
1116    /// <para>
1117    /// <para>Determines whether this instance greater than.
1118    /// </para>
1119    /// <para></para>
1120    /// </summary>
1121    /// <param name="first">
1122    /// <para>The first.</para>
1123    /// <para></para>
1124    /// </param>
1125    /// <param name="second">
1126    /// <para>The second.</para>
1127    /// <para></para>
1128    /// </param>
1129    /// <returns>
1130    /// <para>The bool</para>
1131    /// <para></para>
1132    /// </returns>
1133    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1134    protected virtual bool GreaterThan(TLinkAddress first, TLinkAddress second) =>
1135        ↪ _comparer.Compare(first, second) > 0;
1136
1137    /// <summary>
1138    /// <para>
1139    /// <para>Determines whether this instance greater or equal than.
1140    /// </para>
1141    /// <para></para>
1142    /// </summary>
1143    /// <param name="first">
1144    /// <para>The first.</para>
1145    /// <para></para>
1146    /// </param>
1147    /// <param name="second">
1148    /// <para>The second.</para>
1149    /// <para></para>
1150    /// </param>
1151    /// <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">
1293     /// <para>The was disposed.</para>
1294     /// <para></para>
1295     /// </param>
1296     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1297     protected override void Dispose(bool manual, bool wasDisposed)
1298     {
1299         if (!wasDisposed)
1300         {

```

```

1298         ResetPointers();
1299         _dataMemory.DisposeIfPossible();
1300         _indexMemory.DisposeIfPossible();
1301     }
1302 }
1303
1304 #endregion
1305 }
1306 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Methods.Lists;
3  using Platform.Converters;
4  using static System.Runtime.CompilerServices.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split.Generic
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unused links list methods.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="AbsoluteCircularDoublyLinkedListMethods{TLinkAddress}"/>
17     /// <seealso cref="ILinksListMethods{TLinkAddress}"/>
18     public unsafe class UnusedLinksListMethods<TLinkAddress> :
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         /// <summary>
62         /// <para>
63         /// Gets the link data part reference using the specified link.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         /// <param name="link">
68         /// <para>The link.</para>
69         /// </param>

```

```

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>
        ↪ GetLinkDataPartReference(TLinkAddress link) => ref
        ↪ AsRef<RawLinkDataPart<TLinkAddress>>(_links +
        ↪ (RawLinkDataPart<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) =>
        ↪ GetLinkDataPartReference(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) =>
        ↪ GetLinkDataPartReference(element).Target;
134
135     /// <summary>
136     /// <para>
137     /// Gets the size.

```

```

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

```

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

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link data part.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLinkDataPart<TLinkAddress> : IEquatable<RawLinkDataPart<TLinkAddress>>
17     {
18         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
            ↪ 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)]
59         public override bool Equals(object obj) => obj is RawLinkDataPart<TLinkAddress> link ?
            ↪ Equals(link) : false;
60
61         /// <summary>
62         /// <para>
63         /// Determines whether this instance equals.
64         /// </para>
65         /// <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,
94         ↪ RawLinkDataPart<TLinkAddress> right) => left.Equals(right);
95
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public static bool operator !=(RawLinkDataPart<TLinkAddress> left,
98         ↪ RawLinkDataPart<TLinkAddress> right) => !(left == right);
99 }

```

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

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link index part.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLinkIndexPart<TLinkAddress> : IEquatable<RawLinkIndexPart<TLinkAddress>>
17     {
18         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
19             ↪ EqualityComparer<TLinkAddress>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLinkIndexPart<TLinkAddress>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The root as source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLinkAddress RootAsSource;
36
37         /// <summary>
38         /// <para>
39         /// The left as source.
40         /// </para>
41         /// <para></para>
42         /// </summary>

```



```

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 }

```

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

```

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

```

```

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     /// </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 UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLi
    ↪ nkAddress> 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></para>
77     /// </summary>
78     /// <returns>
79     /// <para>The link</para>
80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override TLinkAddress GetZero() => 0U;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equal to zero.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="value">
92     /// <para>The value.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    protected override bool EqualToZero(TLinkAddress value) => value == 0U;
101
102     /// <summary>
103     /// <para>
104     /// Determines whether this instance are equal.
105     /// </para>
106     /// <para></para>
107     /// </summary>
108     /// <param name="first">
109     /// <para>The first.</para>
110     /// <para></para>
111     /// </param>
112     /// <param name="second">
113     /// <para>The second.</para>
114     /// <para></para>
115     /// </param>
116     /// <returns>
117     /// <para>The bool</para>

```

```

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

```

```

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

```

```

267     /// </param>
268     /// <returns>
269     /// <para>The bool</para>
270     /// <para></para>
271     /// </returns>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override bool LessThan(TLinkAddress first, TLinkAddress second) => first <
        ↪ second;

274
275     /// <summary>
276     /// <para>
277     /// Increments the value.
278     /// </para>
279     /// <para></para>
280     /// </summary>
281     /// <param name="value">
282     /// <para>The value.</para>
283     /// <para></para>
284     /// </param>
285     /// <returns>
286     /// <para>The link</para>
287     /// <para></para>
288     /// </returns>
289     [MethodImpl(MethodImplOptions.AggressiveInlining)]
290     protected override TLinkAddress Increment(TLinkAddress value) => ++value;
291
292     /// <summary>
293     /// <para>
294     /// Decrements the value.
295     /// </para>
296     /// <para></para>
297     /// </summary>
298     /// <param name="value">
299     /// <para>The value.</para>
300     /// <para></para>
301     /// </param>
302     /// <returns>
303     /// <para>The link</para>
304     /// <para></para>
305     /// </returns>
306     [MethodImpl(MethodImplOptions.AggressiveInlining)]
307     protected override TLinkAddress Decrement(TLinkAddress value) => --value;
308
309     /// <summary>
310     /// <para>
311     /// Adds the first.
312     /// </para>
313     /// <para></para>
314     /// </summary>
315     /// <param name="first">
316     /// <para>The first.</para>
317     /// <para></para>
318     /// </param>
319     /// <param name="second">
320     /// <para>The second.</para>
321     /// <para></para>
322     /// </param>
323     /// <returns>
324     /// <para>The link</para>
325     /// <para></para>
326     /// </returns>
327     [MethodImpl(MethodImplOptions.AggressiveInlining)]
328     protected override TLinkAddress Add(TLinkAddress first, TLinkAddress second) => first +
        ↪ second;

329
330     /// <summary>
331     /// <para>
332     /// Subtracts the first.
333     /// </para>
334     /// <para></para>
335     /// </summary>
336     /// <param name="first">
337     /// <para>The first.</para>
338     /// <para></para>
339     /// </param>
340     /// <param name="second">
341     /// <para>The second.</para>
342     /// <para></para>

```

```

343     /// </param>
344     /// <returns>
345     /// <para>The link</para>
346     /// <para></para>
347     /// </returns>
348     [MethodImpl(MethodImplOptions.AggressiveInlining)]
349     protected override TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
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>
413     /// <para></para>
414     /// </param>
415     /// <returns>
416     /// <para>The bool</para>
417     /// <para></para>
418     /// </returns>
419     [MethodImpl(MethodImplOptions.AggressiveInlining)]
420     protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLinkAddress = System.UInt32;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 32 external links size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16    /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
17    public unsafe abstract class UInt32ExternalLinksSizeBalancedTreeMethodsBase :
        ↪ 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="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)]
protected UInt32ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
↳ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
: base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
{
    LinksDataParts = linksDataParts;
    LinksIndexParts = linksIndexParts;
    Header = header;
}
70
71
72 /// <summary>
73 /// <para>
74 /// Gets the zero.
75 /// </para>
76 /// <para></para>
77 /// </summary>
78 /// <returns>
79 /// <para>The link</para>
80 /// <para></para>
81 /// </returns>
82 [MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override TLinkAddress GetZero() => OU;
84
85 /// <summary>
86 /// <para>
87 /// Determines whether this instance equal to zero.
88 /// </para>
89 /// <para></para>
90 /// </summary>
91 /// <param name="value">
92 /// <para>The value.</para>
93 /// <para></para>
94 /// </param>
95 /// <returns>
96 /// <para>The bool</para>
97 /// <para></para>
98 /// </returns>
99 [MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override bool EqualToZero(TLinkAddress value) => value == OU;
101
102 /// <summary>
103 /// <para>
104 /// Determines whether this instance are equal.
105 /// </para>
106 /// <para></para>
107 /// </summary>
108 /// <param name="first">
109 /// <para>The first.</para>
110 /// <para></para>
111 /// </param>
112 /// <param name="second">
113 /// <para>The second.</para>
114 /// <para></para>

```

```

115     /// </param>
116     /// <returns>
117     /// <para>The bool</para>
118     /// <para></para>
119     /// </returns>
120     [MethodImpl(MethodImplOptions.AggressiveInlining)]
121     protected override bool AreEqual(TLinkAddress first, TLinkAddress second) => first ==
        ↪ second;

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

160
161     /// <summary>
162     /// <para>
163     /// Determines whether this instance greater or equal than.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="first">
168     /// <para>The first.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="second">
172     /// <para>The second.</para>
173     /// <para></para>
174     /// </param>
175     /// <returns>
176     /// <para>The bool</para>
177     /// <para></para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected override bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
        ↪ first >= second;

181
182     /// <summary>
183     /// <para>
184     /// Determines whether this instance greater or equal than zero.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="value">
189     /// <para>The value.</para>

```

```

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

```

```

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

```

```

340    /// <param name="second">
341    /// <para>The second.</para>
342    /// <para></para>
343    /// </param>
344    /// <returns>
345    /// <para>The link</para>
346    /// <para></para>
347    /// </returns>
348    [MethodImpl(MethodImplOptions.AggressiveInlining)]
349    protected override TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
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>
413    /// <para></para>
414    /// </param>
415    /// <returns>
416    /// <para>The bool</para>
417    /// <para></para>

```

```

415     /// </returns>
416     [MethodImpl(MethodImplOptions.AggressiveInlining)]
417     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
418     {
419         ref var firstLink = ref LinksDataParts[first];
420         ref var secondLink = ref LinksDataParts[second];
421         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
422             ↪ secondLink.Source, secondLink.Target);
423     }
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance first is to the right of second.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The bool</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
444         ↪ second)
445     {
446         ref var firstLink = ref LinksDataParts[first];
447         ref var secondLink = ref LinksDataParts[second];
448         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
449             ↪ secondLink.Source, secondLink.Target);
450     }
451 }

```

1.52 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods

```

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

```

```

36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
181         ↪ LinksIndexParts[node].SizeAsSource = 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->RootAsSource;
190
191     /// <summary>
192     /// <para>
193     /// Gets the base part value using the specified node.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="node">
198     /// <para>The node.</para>
199     /// <para></para>
200     /// </param>
201     /// <returns>
202     /// <para>The link</para>
203     /// <para></para>
204     /// </returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
207         ↳ LinksDataParts[node].Source;
208
209     /// <summary>
210     /// <para>
211     /// Determines whether this instance first is to the left of second.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="firstSource">
216     /// <para>The first source.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="firstTarget">
220     /// <para>The first target.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="secondSource">
224     /// <para>The second source.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="secondTarget">
228     /// <para>The second target.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>The bool</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
237         ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
238         => firstSource < secondSource || firstSource == secondSource && firstTarget <
239         ↳ secondTarget;
240
241     /// <summary>
242     /// <para>
243     /// Determines whether this instance first is to the right of second.
244     /// </para>
245     /// <para></para>
246     /// </summary>
247     /// <param name="firstSource">
248     /// <para>The first source.</para>
249     /// <para></para>
250     /// </param>
251     /// <param name="firstTarget">
252     /// <para>The first target.</para>
253     /// <para></para>
254     /// </param>
255     /// <param name="secondSource">
256     /// <para>The second source.</para>
257     /// <para></para>
258     /// </param>
259     /// <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         => firstSource > secondSource || firstSource == secondSource && firstTarget >
271         ↳ secondTarget;

```

```

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 }

```

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

```

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"/>
    ↪ 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 UInt32ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
    ↳ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    ↳ : base(constants, linksDataParts, linksIndexParts, header) { }

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

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

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

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

109
110 /// <summary>

```

```

111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
126        ↪ LinksIndexParts[node].LeftAsSource = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
144        ↪ LinksIndexParts[node].RightAsSource = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>
154    /// <para></para>
155    /// </param>
156    /// <returns>
157    /// <para>The link</para>
158    /// <para></para>
159    /// </returns>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
161    protected override TLinkAddress GetSize(TLinkAddress node) =>
162        ↪ LinksIndexParts[node].SizeAsSource;
163
164    /// <summary>
165    /// <para>
166    /// Sets the size using the specified node.
167    /// </para>
168    /// <para></para>
169    /// </summary>
170    /// <param name="node">
171    /// <para>The node.</para>
172    /// <para></para>
173    /// </param>
174    /// <param name="size">
175    /// <para>The size.</para>
176    /// <para></para>
177    /// </param>
178    [MethodImpl(MethodImplOptions.AggressiveInlining)]
179    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
180        ↪ LinksIndexParts[node].SizeAsSource = size;
181
182    /// <summary>
183    /// <para>
184    /// Gets the tree root.
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->RootAsSource;
190
191     /// <summary>
192     /// <para>
193     /// Gets the base part value using the specified node.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="node">
198     /// <para>The node.</para>
199     /// <para></para>
200     /// </param>
201     /// <returns>
202     /// <para>The link</para>
203     /// <para></para>
204     /// </returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
207         ↪ LinksDataParts[node].Source;
208
209     /// <summary>
210     /// <para>
211     /// Determines whether this instance first is to the left of second.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="firstSource">
216     /// <para>The first source.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="firstTarget">
220     /// <para>The first target.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="secondSource">
224     /// <para>The second source.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="secondTarget">
228     /// <para>The second target.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>The bool</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
237         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
238         => firstSource < secondSource || firstSource == secondSource && firstTarget <
239         ↪ secondTarget;
240
241     /// <summary>
242     /// <para>
243     /// Determines whether this instance first is to the right of second.
244     /// </para>
245     /// <para></para>
246     /// </summary>
247     /// <param name="firstSource">
248     /// <para>The first source.</para>
249     /// <para></para>
250     /// </param>
251     /// <param name="firstTarget">
252     /// <para>The first target.</para>
253     /// <para></para>
254     /// </param>
255     /// <param name="secondSource">
256     /// <para>The second source.</para>
257     /// <para></para>
258     /// </param>
259     /// <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         => firstSource > secondSource || firstSource == secondSource && firstTarget >
271         ↪ secondTarget;

```

```

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

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

```

1.54 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/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></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></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 UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
    ↪ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
41
42     /// <summary>

```

```

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

```

```

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

```



```

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

```

```
protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
    => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
        ↳ secondSource;

/// <summary>
/// <para>
/// Clears the node using the specified node.
/// </para>
/// <para></para>
/// </summary>
/// <param name="node">
/// <para>The node.</para>
/// <para></para>
/// </param>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override void ClearNode(TLinkAddress node)
{
    ref var link = ref LinksIndexParts[node];
    link.LeftAsTarget = Zero;
    link.RightAsTarget = Zero;
    link.SizeAsTarget = Zero;
}
```

1.55 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethod

```

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 :
16    ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see cref="UInt32ExternalLinksTargetsSizeBalancedTreeMethods"/>
21        ↪ instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public UInt32ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<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>

```

```

48     /// <param name="node">
49     /// <para>The node.</para>
50     /// </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     /// </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     /// <para></para>
150     /// </summary>
151     /// <param name="node">
152     /// <para>The node.</para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override 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     /// </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.
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->RootAsTarget;
191
192     /// <summary>
193     /// <para>
194     /// Gets the base part value using the specified node.
195     /// </para>
196     /// <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     /// <para>Determines whether this instance first is to the left of second.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="firstSource">
216     /// <para>The first source.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="firstTarget">
220     /// <para>The first target.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="secondSource">
224     /// <para>The second source.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="secondTarget">
228     /// <para>The second target.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>The bool</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
237         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
238         => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
239         ↪ secondSource;
240
241     /// <summary>
242     /// <para>
243     /// <para>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;

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLinkAddress = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 internal links recursionless size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16     public unsafe abstract class UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
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="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="constants">
49         /// <para>A constants.</para>
50         /// <para></para>
51         /// </param>
52         /// <param name="linksDataParts">
53         /// <para>A links data parts.</para>
54         /// <para></para>
55         /// </param>
56         /// <param name="linksIndexParts">
57         /// <para>A links index parts.</para>
58         /// <para></para>
59         /// </param>

```

```

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 UInt32 InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        ↳ : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64     {
65         LinksDataParts = linksDataParts;
66         LinksIndexParts = linksIndexParts;
67         Header = header;
68     }
69
70
71     /// <summary>
72     /// <para>
73     /// Gets the zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <returns>
78     /// <para>The link</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override TLinkAddress GetZero() => 0U;
83
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance equal to zero.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override bool EqualToZero(TLinkAddress value) => value == 0U;
100
101     /// <summary>
102     /// <para>
103     /// Determines whether this instance are equal.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="first">
108     /// <para>The first.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="second">
112     /// <para>The second.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The bool</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override bool AreEqual(TLinkAddress first, TLinkAddress second) => first ==
        ↳ second;
121
122     /// <summary>
123     /// <para>
124     /// Determines whether this instance greater than zero.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="value">
129     /// <para>The value.</para>
130     /// <para></para>

```

```

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

```



```

206     /// <para></para>
207     /// </param>
208     /// <returns>
209     /// <para>The bool</para>
210     /// <para></para>
211     /// </returns>
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]
213     protected override bool LessOrEqualThanZero(TLinkAddress value) => value == 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(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     /// <summary>
367     /// <para>
368     /// Gets the link index part reference using the specified link.
369     /// </para>
370     /// <para></para>
371     /// </summary>
372     /// <param name="link">
373     /// <para>The link.</para>
374     /// <para></para>
375     /// </param>
376     /// <returns>
377     /// <para>A ref raw link index part of t link</para>
378     /// <para></para>
379     /// </returns>
380     [MethodImpl(MethodImplOptions.AggressiveInlining)]
381     protected override ref RawLinkIndexPart<TLinkAddress>
        ↳ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];

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

402     /// <summary>
403     /// <para>
404     /// Determines whether this instance first is to the right of second.
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 bool</para>
418     /// <para></para>
419     /// </returns>
420     [MethodImpl(MethodImplOptions.AggressiveInlining)]
421     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
        ↳ second) => GetKeyPartValue(first) > GetKeyPartValue(second);

```

```

    }

```

```

}

```

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

```

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

```

```

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

```

```

152     /// </param>
153     /// <returns>
154     /// <para>The bool</para>
155     /// <para></para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override bool GreaterThan(TLinkAddress first, TLinkAddress second) => first >
        ↪ second;

159
160     /// <summary>
161     /// <para>
162     /// Determines whether this instance greater or equal than.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     /// <param name="first">
167     /// <para>The first.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="second">
171     /// <para>The second.</para>
172     /// <para></para>
173     /// </param>
174     /// <returns>
175     /// <para>The bool</para>
176     /// <para></para>
177     /// </returns>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
179     protected override bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
        ↪ first >= second;

180
181     /// <summary>
182     /// <para>
183     /// Determines whether this instance greater or equal than zero.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="value">
188     /// <para>The value.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The bool</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override bool GreaterOrEqualThanZero(TLinkAddress value) => true; // value >=
        ↪ 0 is always true for ulong

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

214
215     /// <summary>
216     /// <para>
217     /// Determines whether this instance less or equal than.
218     /// </para>
219     /// <para></para>
220     /// </summary>
221     /// <param name="first">
222     /// <para>The first.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="second">

```

```

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

```

```

301    /// <returns>
302    /// <para>The link</para>
303    /// <para></para>
304    /// </returns>
305    [MethodImpl(MethodImplOptions.AggressiveInlining)]
306    protected override TLinkAddress Decrement(TLinkAddress value) => --value;
307
308    /// <summary>
309    /// <para>
310    /// Adds the first.
311    /// </para>
312    /// <para></para>
313    /// </summary>
314    /// <param name="first">
315    /// <para>The first.</para>
316    /// <para></para>
317    /// </param>
318    /// <param name="second">
319    /// <para>The second.</para>
320    /// <para></para>
321    /// </param>
322    /// <returns>
323    /// <para>The link</para>
324    /// <para></para>
325    /// </returns>
326    [MethodImpl(MethodImplOptions.AggressiveInlining)]
327    protected override TLinkAddress Add(TLinkAddress first, TLinkAddress second) => first +
    ↪ second;
328
329    /// <summary>
330    /// <para>
331    /// Subtracts the first.
332    /// </para>
333    /// <para></para>
334    /// </summary>
335    /// <param name="first">
336    /// <para>The first.</para>
337    /// <para></para>
338    /// </param>
339    /// <param name="second">
340    /// <para>The second.</para>
341    /// <para></para>
342    /// </param>
343    /// <returns>
344    /// <para>The link</para>
345    /// <para></para>
346    /// </returns>
347    [MethodImpl(MethodImplOptions.AggressiveInlining)]
348    protected override TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
    ↪ first - second;
349
350    /// <summary>
351    /// <para>
352    /// Gets the link data part reference using the specified link.
353    /// </para>
354    /// <para></para>
355    /// </summary>
356    /// <param name="link">
357    /// <para>The link.</para>
358    /// <para></para>
359    /// </param>
360    /// <returns>
361    /// <para>A ref raw link data part of t link</para>
362    /// <para></para>
363    /// </returns>
364    [MethodImpl(MethodImplOptions.AggressiveInlining)]
365    protected override ref RawLinkDataPart<TLinkAddress>
    ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
366
367    /// <summary>
368    /// <para>
369    /// Gets the link index part reference using the specified link.
370    /// </para>
371    /// <para></para>
372    /// </summary>
373    /// <param name="link">
374    /// <para>The link.</para>
375    /// <para></para>

```



```

376     /// </param>
377     /// <returns>
378     /// <para>A ref raw link index part of t link</para>
379     /// <para></para>
380     /// </returns>
381     [MethodImpl(MethodImplOptions.AggressiveInlining)]
382     protected override ref RawLinkIndexPart<TLinkAddress>
        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
383
384     /// <summary>
385     /// <para>
386     /// Determines whether this instance first is to the left of second.
387     /// </para>
388     /// <para></para>
389     /// </summary>
390     /// <param name="first">
391     /// <para>The first.</para>
392     /// <para></para>
393     /// </param>
394     /// <param name="second">
395     /// <para>The second.</para>
396     /// <para></para>
397     /// </param>
398     /// <returns>
399     /// <para>The bool</para>
400     /// <para></para>
401     /// </returns>
402     [MethodImpl(MethodImplOptions.AggressiveInlining)]
403     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress
        ↪ second) => GetKeyPartValue(first) < GetKeyPartValue(second);
404
405     /// <summary>
406     /// <para>
407     /// Determines whether this instance first is to the right of second.
408     /// </para>
409     /// <para></para>
410     /// </summary>
411     /// <param name="first">
412     /// <para>The first.</para>
413     /// <para></para>
414     /// </param>
415     /// <param name="second">
416     /// <para>The second.</para>
417     /// <para></para>
418     /// </param>
419     /// <returns>
420     /// <para>The bool</para>
421     /// <para></para>
422     /// </returns>
423     [MethodImpl(MethodImplOptions.AggressiveInlining)]
424     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
        ↪ second) => GetKeyPartValue(first) > GetKeyPartValue(second);
425 }
426 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Generic
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources linked list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLinkAddress}"/>
15     public unsafe class UInt32InternalLinksSourcesLinkedListMethods :
        ↪ InternalLinksSourcesLinkedListMethods<TLinkAddress>
16     {
17         private readonly RawLinkDataPart<TLinkAddress>* _linksDataParts;
18         private readonly RawLinkIndexPart<TLinkAddress>* _linksIndexParts;
19
20         /// <summary>
21         /// <para>

```

```

22     /// Initializes a new <see cref="UInt32InternalLinksSourcesLinkedListMethods"/> instance.
23     /// </para>
24     /// </summary>
25     /// <param name="constants">
26     /// <para>A constants.</para>
27     /// </param>
28     /// <param name="linksDataParts">
29     /// <para>A links data parts.</para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress>
36     ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
37     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts)
38     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
39     {
40         _linksDataParts = linksDataParts;
41         _linksIndexParts = linksIndexParts;
42     }
43
44     /// <summary>
45     /// <para>
46     /// Gets the link data part reference using the specified link.
47     /// </para>
48     /// </summary>
49     /// <param name="link">
50     /// <para>The link.</para>
51     /// </param>
52     /// <returns>
53     /// <para>A ref raw link data part of t link</para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref RawLinkDataPart<TLinkAddress>
57     ↪ GetLinkDataPartReference(TLinkAddress link) => ref _linksDataParts[link];
58
59     /// <summary>
60     /// <para>
61     /// Gets the link index part reference using the specified link.
62     /// </para>
63     /// </summary>
64     /// <param name="link">
65     /// <para>The link.</para>
66     /// </param>
67     /// <returns>
68     /// <para>A ref raw link index part of t link</para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref RawLinkIndexPart<TLinkAddress>
72     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref _linksIndexParts[link];
73
74     }
75 }
76
77
78
79
80

```

1.59 ./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     /// </summary>
13

```

```

14  /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15  public unsafe class UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
    ↳ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16  {
17      /// <summary>
18      /// <para>
19      ///     Initializes a new <see
    ↳ cref="UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
20      /// </para>
21      /// <para></para>
22      /// </summary>
23      /// <param name="constants">
24      /// <para>A constants.</para>
25      /// <para></para>
26      /// </param>
27      /// <param name="linksDataParts">
28      /// <para>A links data parts.</para>
29      /// <para></para>
30      /// </param>
31      /// <param name="linksIndexParts">
32      /// <para>A links index parts.</para>
33      /// <para></para>
34      /// </param>
35      /// <param name="header">
36      /// <para>A header.</para>
37      /// <para></para>
38      /// </param>
39      [MethodImpl(MethodImplOptions.AggressiveInlining)]
40      public UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<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) =>
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">
155    /// <para>The node.</para>
156    /// <para></para>
157    /// </param>
158    /// <returns>
159    /// <para>The link</para>
160    /// <para></para>
161    /// </returns>
162    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

159     protected override TLinkAddress GetSize(TLinkAddress node) =>
160         ↪ LinksIndexParts[node].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     /// </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         ↪ LinksIndexParts[node].SizeAsSource = size;
176
177     /// <summary>
178     /// <para>
179     /// Gets the tree root using the specified node.
180     /// </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     /// <summary>
193     /// <para>
194     /// Gets the base part value using the specified node.
195     /// </para>
196     /// </summary>
197     /// <param name="node">
198     /// <para>The node.</para>
199     /// </param>
200     /// <returns>
201     /// <para>The link</para>
202     /// </returns>
203     [MethodImpl(MethodImplOptions.AggressiveInlining)]
204     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
205         ↪ LinksDataParts[node].Source;
206
207     /// <summary>
208     /// <para>
209     /// Gets the key part value using the specified node.
210     /// </para>
211     /// </summary>
212     /// <param name="node">
213     /// <para>The node.</para>
214     /// </param>
215     /// <returns>
216     /// <para>The link</para>
217     /// </returns>
218     [MethodImpl(MethodImplOptions.AggressiveInlining)]
219     protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
220         ↪ LinksDataParts[node].Target;
221
222     /// <summary>
223     /// <para>
224

```

```

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

```

1.60 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksSourcesSizeBalancedTreeMethods :
16         ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32InternalLinksSourcesSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">

```

```

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

```

```

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 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using TLinkAddress = System.UInt32;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 32 internal links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
        ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
42
43         /// <summary>
44         /// <para>
45         /// Gets the left reference using the specified node.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <param name="node">
50         /// <para>The node.</para>
51         /// <para></para>
52         /// </param>
53         /// <returns>
54         /// <para>The ref link</para>
55         /// <para></para>
56         /// </returns>
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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)]
129    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
130        ↪ LinksIndexParts[node].LeftAsTarget = left;
131
132    /// <summary>
133    /// <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     /// <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].SizeAsTarget;
162
163     /// <summary>
164     /// <para>
165     /// Sets the size using the specified node.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="node">
170     /// <para>The node.</para>
171     /// <para></para>
172     /// </param>
173     /// <param name="size">
174     /// <para>The size.</para>
175     /// <para></para>
176     /// </param>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
179         ↪ LinksIndexParts[node].SizeAsTarget = size;
180
181     /// <summary>
182     /// <para>
183     /// Gets the tree root using the specified node.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="node">
188     /// <para>The node.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The link</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override TLinkAddress GetTreeRoot(TLinkAddress node) =>
197         ↪ LinksIndexParts[node].RootAsTarget;
198
199     /// <summary>
200     /// <para>
201     /// Gets the base part value using the specified node.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="node">
206     /// <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);
270 }

```

1.62 ./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 :
    ↳ UInt32InternalLinksSizeBalancedTreeMethodsBase
16 {
17     /// <summary>
18     /// <para>
19     /// Initializes a new <see cref="UInt32InternalLinksTargetsSizeBalancedTreeMethods"/>
    ↳ instance.
20     /// </para>
21     /// <para></para>
22     /// </summary>
23     /// <param name="constants">
24     /// <para>A constants.</para>
25     /// <para></para>
26     /// </param>
27     /// <param name="linksDataParts">
28     /// <para>A links data parts.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt32InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
    ↳ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    ↳ : base(constants, linksDataParts, linksIndexParts, header) { }
41
42     /// <summary>
43     /// <para>
44     /// Gets the left reference using the specified node.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="node">
49     /// <para>The node.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
    ↳ LinksIndexParts[node].LeftAsTarget;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↳ LinksIndexParts[node].RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.

```

```

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

```

```

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>
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>
230     [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 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLinkAddress = System.UInt32;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the int 32 split memory links.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="SplitMemoryLinksBase{TLinkAddress}"/>
19     public unsafe class UInt32SplitMemoryLinks : SplitMemoryLinksBase<TLinkAddress>
20     {
21         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalSourceTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalTargetTreeMethods;
24         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalTargetTreeMethods;
25         private LinksHeader<TLinkAddress>* _header;
26         private RawLinkDataPart<TLinkAddress>* _linksDataParts;
27         private RawLinkIndexPart<TLinkAddress>* _linksIndexParts;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
32         /// </para>
33         /// <para></para>

```

```

34     /// </summary>
35     /// <param name="dataMemory">
36     /// <para>A data memory.</para>
37     /// <para></para>
38     /// </param>
39     /// <param name="indexMemory">
40     /// <para>A index memory.</para>
41     /// <para></para>
42     /// </param>
43     [MethodImpl(MethodImplOptions.AggressiveInlining)]
44     public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46     /// <summary>
47     /// <para>
48     /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="dataMemory">
53     /// <para>A data memory.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="indexMemory">
57     /// <para>A index memory.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="memoryReservationStep">
61     /// <para>A memory reservation step.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↳ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
        ↳ IndexTreeType.Default, useLinkedList: true) { }
66
67     /// <summary>
68     /// <para>
69     /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
70     /// </para>
71     /// <para></para>
72     /// </summary>
73     /// <param name="dataMemory">
74     /// <para>A data memory.</para>
75     /// <para></para>
76     /// </param>
77     /// <param name="indexMemory">
78     /// <para>A index memory.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="memoryReservationStep">
82     /// <para>A memory reservation step.</para>
83     /// <para></para>
84     /// </param>
85     /// <param name="constants">
86     /// <para>A constants.</para>
87     /// <para></para>
88     /// </param>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep, LinksConstants<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,
        ↪ IResizableDirectMemory indexMemory)
158     {
159         _linksDataParts = (RawLinkDataPart<TLinkAddress>*)dataMemory.Pointer;
160         _linksIndexParts = (RawLinkIndexPart<TLinkAddress>*)indexMemory.Pointer;

```

```

161     _header = (LinksHeader<TLinkAddress>*)indexMemory.Pointer;
162     if (_useLinkedList)
163     {
164         InternalSourcesListMethods = new
            ↳ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
            ↳ _linksIndexParts);
165     }
166     else
167     {
168         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
169     }
170     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
171     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
172     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
173     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
174 }
175
176 /// <summary>
177 /// <para>
178 /// Resets the pointers.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 [MethodImpl(MethodImplOptions.AggressiveInlining)]
183 protected override void ResetPointers()
184 {
185     base.ResetPointers();
186     _linksDataParts = null;
187     _linksIndexParts = null;
188     _header = null;
189 }
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>
    ↳ GetLinkDataPartReference(TLinkAddress linkIndex) => ref _linksDataParts[linkIndex];
220
221 /// <summary>
222 /// <para>
223 /// Gets the link index part reference using the specified link index.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="linkIndex">
228 /// <para>The link index.</para>
229 /// <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     /// <summary>
322     /// <para>
323     /// Determines whether this instance greater or equal than.
324     /// </para>
325     /// <para></para>
326     /// </summary>
327     /// <param name="first">
328     /// <para>The first.</para>
329     /// <para></para>
330     /// </param>
331     /// <param name="second">
332     /// <para>The second.</para>
333     /// <para></para>
334     /// </param>
335     /// <returns>
336     /// <para>The bool</para>
337     /// <para></para>
338     /// </returns>
339     [MethodImpl(MethodImplOptions.AggressiveInlining)]
340     protected override bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
        ↪ first >= second;

342     /// <summary>
343     /// <para>
344     /// Gets the zero.
345     /// </para>
346     /// <para></para>
347     /// </summary>
348     /// <returns>
349     /// <para>The link</para>
350     /// <para></para>
351     /// </returns>
352     [MethodImpl(MethodImplOptions.AggressiveInlining)]
353     protected override TLinkAddress GetZero() => 0U;

355     /// <summary>
356     /// <para>
357     /// Gets the one.
358     /// </para>
359     /// <para></para>
360     /// </summary>
361     /// <returns>
362     /// <para>The link</para>
363     /// <para></para>
364     /// </returns>
365     [MethodImpl(MethodImplOptions.AggressiveInlining)]
366     protected override TLinkAddress GetOne() => 1U;

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(TLinkAddress value) => value;
384
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>
456     /// </param>
457     /// <returns>
458     /// <para>The link</para>
459     /// <para></para>
460     /// </returns>
461     [MethodImpl(MethodImplOptions.AggressiveInlining)]
462     protected override TLinkAddress Increment(TLinkAddress link) => ++link;

```

```

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 }

```

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

```

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

```



```

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

```

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

```

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

```

```

58     /// </param>
59     /// <param name="header">
60     /// <para>A header.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected UInt64 ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↳ 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></para>
77     /// </summary>
78     /// <returns>
79     /// <para>The ulong</para>
80     /// <para></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></para>
90     /// </summary>
91     /// <param name="value">
92     /// <para>The value.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    protected override bool EqualToZero(ulong value) => value == 0UL;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance are equal.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="first">
109    /// <para>The first.</para>
110    /// <para></para>
111    /// </param>
112    /// <param name="second">
113    /// <para>The second.</para>
114    /// <para></para>
115    /// </param>
116    /// <returns>
117    /// <para>The bool</para>
118    /// <para></para>
119    /// </returns>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    protected override bool AreEqual(ulong first, ulong second) => first == second;
122
123    /// <summary>
124    /// <para>
125    /// Determines whether this instance greater than zero.
126    /// </para>
127    /// <para></para>
128    /// </summary>
129    /// <param name="value">
130    /// <para>The value.</para>
131    /// <para></para>
132    /// </param>
133    /// </returns>

```

```

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

```

```

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

```

```

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

```

```

365 /// <para>
366 /// Gets the link data part reference using the specified link.
367 /// </para>
368 /// <para></para>
369 /// </summary>
370 /// <param name="link">
371 /// <para>The link.</para>
372 /// <para></para>
373 /// </param>
374 /// <returns>
375 /// <para>A ref raw link data part of t link</para>
376 /// <para></para>
377 /// </returns>
378 [MethodImpl(MethodImplOptions.AggressiveInlining)]
379 protected override ref RawLinkDataPart<TLinkAddress>
    ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];

380
381 /// <summary>
382 /// <para>
383 /// Gets the link index part reference using the specified link.
384 /// </para>
385 /// <para></para>
386 /// </summary>
387 /// <param name="link">
388 /// <para>The link.</para>
389 /// <para></para>
390 /// </param>
391 /// <returns>
392 /// <para>A ref raw link index part of t link</para>
393 /// <para></para>
394 /// </returns>
395 [MethodImpl(MethodImplOptions.AggressiveInlining)]
396 protected override ref RawLinkIndexPart<TLinkAddress>
    ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];

397
398 /// <summary>
399 /// <para>
400 /// Determines whether this instance first is to the left of second.
401 /// </para>
402 /// <para></para>
403 /// </summary>
404 /// <param name="first">
405 /// <para>The first.</para>
406 /// <para></para>
407 /// </param>
408 /// <param name="second">
409 /// <para>The second.</para>
410 /// <para></para>
411 /// </param>
412 /// <returns>
413 /// <para>The bool</para>
414 /// <para></para>
415 /// </returns>
416 [MethodImpl(MethodImplOptions.AggressiveInlining)]
417 protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
418 {
419     ref var firstLink = ref LinksDataParts[first];
420     ref var secondLink = ref LinksDataParts[second];
421     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
422 }

423
424 /// <summary>
425 /// <para>
426 /// Determines whether this instance first is to the right of second.
427 /// </para>
428 /// <para></para>
429 /// </summary>
430 /// <param name="first">
431 /// <para>The first.</para>
432 /// <para></para>
433 /// </param>
434 /// <param name="second">
435 /// <para>The second.</para>
436 /// <para></para>
437 /// </param>
438 /// <returns>
439 /// <para>The bool</para>

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLinkAddress = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 external links size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
17     public unsafe abstract class UInt64ExternalLinksSizeBalancedTreeMethodsBase :
    ↪ 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></para>
58         /// </param>
59         /// <param name="header">
60         /// <para>A header.</para>
61         /// <para></para>

```

```

62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected UInt64ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
        ↳ 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></para>
77     /// </summary>
78     /// <returns>
79     /// <para>The ulong</para>
80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override ulong GetZero() => OUL;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equal to zero.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="value">
92     /// <para>The value.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    protected override bool EqualToZero(ulong value) => value == OUL;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance are equal.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="first">
109    /// <para>The first.</para>
110    /// <para></para>
111    /// </param>
112    /// <param name="second">
113    /// <para>The second.</para>
114    /// <para></para>
115    /// </param>
116    /// <returns>
117    /// <para>The bool</para>
118    /// <para></para>
119    /// </returns>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    protected override bool AreEqual(ulong first, ulong second) => first == second;
122
123    /// <summary>
124    /// <para>
125    /// Determines whether this instance greater than zero.
126    /// </para>
127    /// <para></para>
128    /// </summary>
129    /// <param name="value">
130    /// <para>The value.</para>
131    /// <para></para>
132    /// </param>
133    /// <returns>
134    /// <para>The bool</para>
135    /// <para></para>
136    /// </returns>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

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

```

```

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

```

```

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

```

```

368    /// <para></para>
369    /// </summary>
370    /// <param name="link">
371    /// <para>The link.</para>
372    /// <para></para>
373    /// </param>
374    /// <returns>
375    /// <para>A ref raw link data part of t link</para>
376    /// <para></para>
377    /// </returns>
378    [MethodImpl(MethodImplOptions.AggressiveInlining)]
379    protected override ref RawLinkDataPart<TLinkAddress>
        ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];

380
381    /// <summary>
382    /// <para>
383    /// Gets the link index part reference using the specified link.
384    /// </para>
385    /// <para></para>
386    /// </summary>
387    /// <param name="link">
388    /// <para>The link.</para>
389    /// <para></para>
390    /// </param>
391    /// <returns>
392    /// <para>A ref raw link index part of t link</para>
393    /// <para></para>
394    /// </returns>
395    [MethodImpl(MethodImplOptions.AggressiveInlining)]
396    protected override ref RawLinkIndexPart<TLinkAddress>
        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];

397
398    /// <summary>
399    /// <para>
400    /// Determines whether this instance first is to the left of second.
401    /// </para>
402    /// <para></para>
403    /// </summary>
404    /// <param name="first">
405    /// <para>The first.</para>
406    /// <para></para>
407    /// </param>
408    /// <param name="second">
409    /// <para>The second.</para>
410    /// <para></para>
411    /// </param>
412    /// <returns>
413    /// <para>The bool</para>
414    /// <para></para>
415    /// </returns>
416    [MethodImpl(MethodImplOptions.AggressiveInlining)]
417    protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
418    {
419        ref var firstLink = ref LinksDataParts[first];
420        ref var secondLink = ref LinksDataParts[second];
421        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
422    }

423
424    /// <summary>
425    /// <para>
426    /// Determines whether this instance first is to the right of second.
427    /// </para>
428    /// <para></para>
429    /// </summary>
430    /// <param name="first">
431    /// <para>The first.</para>
432    /// <para></para>
433    /// </param>
434    /// <param name="second">
435    /// <para>The second.</para>
436    /// <para></para>
437    /// </param>
438    /// <returns>
439    /// <para>The bool</para>
440    /// <para></para>
441    /// </returns>
442    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

1.67 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods

```

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
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>
137    /// <param name="node">
138    /// <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;
211
212     /// <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 }

```

1.68 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMeth

```

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;
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">

```

```

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     /// </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     /// <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         ↳ LinksIndexParts[node].SizeAsSource = size;
177
178     /// <summary>
179     /// <para>
180     /// Gets the tree root.
181     /// </para>
182     /// <para></para>
183     /// </summary>
184     /// <returns>
185     /// <para>The link</para>
186     /// </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     /// </returns>
203     [MethodImpl(MethodImplOptions.AggressiveInlining)]
204     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
205         ↳ LinksDataParts[node].Source;
206
207     /// <summary>
208     /// <para>
209     /// Determines whether this instance first is to the left of second.
210     /// </para>
211     /// <para></para>
212     /// </summary>
213

```

```

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>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
268     ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
269     ↪ => firstSource > secondSource || firstSource == secondSource && firstTarget >
270     ↪ secondTarget;
271
272     /// <summary>
273     /// <para>
274     /// Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override void ClearNode(TLinkAddress node)
284     {
285         ref var link = ref LinksIndexParts[node];
286         link.LeftAsSource = Zero;
287         link.RightAsSource = Zero;
288         link.SizeAsSource = Zero;
289     }
290 }
291 }

```

1.69 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalanced

```

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         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// </param>
34         /// <param name="header">
35         /// <para>A header.</para>
36         /// </param>
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
39             ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
40             ↪ : 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         /// </param>
51         /// <returns>
52         /// <para>The ref link</para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
56             ↪ LinksIndexParts[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         /// </param>
67         /// <returns>
68         /// <para>The ref link</para>
69         /// </returns>
70         /// <para></para>
71     }

```

```

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.
183    /// </para>
184    /// <para></para>
185    /// </summary>
186    /// <returns>
187    /// <para>The link</para>
188    /// <para></para>
189    /// </returns>
190    [MethodImpl(MethodImplOptions.AggressiveInlining)]
191    protected override TLinkAddress GetTreeRoot() => Header->RootAsTarget;
192
193    /// <summary>
194    /// <para>
195    /// Gets the base part value using the specified node.
196    /// </para>
197    /// <para></para>
198    /// </summary>
199    /// <param name="node">
200    /// <para>The node.</para>
201    /// <para></para>
202    /// </param>
203    /// <returns>
204    /// <para>The link</para>
205    /// <para></para>
206    /// </returns>
207    [MethodImpl(MethodImplOptions.AggressiveInlining)]
208    protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
209        ↪ LinksDataParts[node].Target;
210
211    /// <summary>
212    /// <para>
213    /// Determines whether this instance first is to the left of second.
214    /// </para>
215    /// <para></para>
216    /// </summary>
217    /// <param name="firstSource">
218    /// <para>The first source.</para>
219    /// <para></para>
220    /// </param>
221    /// <param name="firstTarget">

```

```

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     /// <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;

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

```

1.70 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethod

```

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>
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].RightAsTarget;
82    }
83 }

```

```

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

```

```

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>
221     /// <param name="firstTarget">
222     /// <para>The first target.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="secondSource">
226     /// <para>The second source.</para>
227     /// <para></para>
228     /// </param>

```

```

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

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

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLinkAddress = System.UInt64;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {

```

```

9  /// <summary>
10  /// <para>
11  /// Represents the int 64 internal links recursionless size balanced tree methods base.
12  /// </para>
13  /// <para></para>
14  /// </summary>
15  /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16  public unsafe abstract class UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
    ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
17  {
18  /// <summary>
19  /// <para>
20  /// The links data parts.
21  /// </para>
22  /// <para></para>
23  /// </summary>
24  protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
25  /// <summary>
26  /// <para>
27  /// The links index parts.
28  /// </para>
29  /// <para></para>
30  /// </summary>
31  protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
32  /// <summary>
33  /// <para>
34  /// The header.
35  /// </para>
36  /// <para></para>
37  /// </summary>
38  protected new readonly LinksHeader<TLinkAddress>* Header;
39
40  /// <summary>
41  /// <para>
42  /// Initializes a new <see
    ↳ cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
43  /// </para>
44  /// <para></para>
45  /// </summary>
46  /// <param name="constants">
47  /// <para>A constants.</para>
48  /// <para></para>
49  /// </param>
50  /// <param name="linksDataParts">
51  /// <para>A links data parts.</para>
52  /// <para></para>
53  /// </param>
54  /// <param name="linksIndexParts">
55  /// <para>A links index parts.</para>
56  /// <para></para>
57  /// </param>
58  /// <param name="header">
59  /// <para>A header.</para>
60  /// <para></para>
61  /// </param>
62  [MethodImpl(MethodImplOptions.AggressiveInlining)]
63  protected UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64  {
65  LinksDataParts = linksDataParts;
66  LinksIndexParts = linksIndexParts;
67  Header = header;
68  }
69
70
71  /// <summary>
72  /// <para>
73  /// Gets the zero.
74  /// </para>
75  /// <para></para>
76  /// </summary>
77  /// <returns>
78  /// <para>The ulong</para>
79  /// <para></para>
80  /// </returns>
81  [MethodImpl(MethodImplOptions.AggressiveInlining)]
82  protected override ulong GetZero() => 0UL;

```

```

83
84    /// <summary>
85    /// <para>
86    /// Determines whether this instance equal to zero.
87    /// </para>
88    /// <para></para>
89    /// </summary>
90    /// <param name="value">
91    /// <para>The value.</para>
92    /// <para></para>
93    /// </param>
94    /// <returns>
95    /// <para>The bool</para>
96    /// <para></para>
97    /// </returns>
98    [MethodImpl(MethodImplOptions.AggressiveInlining)]
99    protected override bool EqualToZero(ulong value) => value == 0UL;
100
101    /// <summary>
102    /// <para>
103    /// Determines whether this instance are equal.
104    /// </para>
105    /// <para></para>
106    /// </summary>
107    /// <param name="first">
108    /// <para>The first.</para>
109    /// <para></para>
110    /// </param>
111    /// <param name="second">
112    /// <para>The second.</para>
113    /// <para></para>
114    /// </param>
115    /// <returns>
116    /// <para>The bool</para>
117    /// <para></para>
118    /// </returns>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override bool AreEqual(ulong first, ulong second) => first == second;
121
122    /// <summary>
123    /// <para>
124    /// Determines whether this instance greater than zero.
125    /// </para>
126    /// <para></para>
127    /// </summary>
128    /// <param name="value">
129    /// <para>The value.</para>
130    /// <para></para>
131    /// </param>
132    /// <returns>
133    /// <para>The bool</para>
134    /// <para></para>
135    /// </returns>
136    [MethodImpl(MethodImplOptions.AggressiveInlining)]
137    protected override bool GreaterThanZero(ulong value) => value > 0UL;
138
139    /// <summary>
140    /// <para>
141    /// Determines whether this instance greater than.
142    /// </para>
143    /// <para></para>
144    /// </summary>
145    /// <param name="first">
146    /// <para>The first.</para>
147    /// <para></para>
148    /// </param>
149    /// <param name="second">
150    /// <para>The second.</para>
151    /// <para></para>
152    /// </param>
153    /// <returns>
154    /// <para>The bool</para>
155    /// <para></para>
156    /// </returns>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override bool GreaterThan(ulong first, ulong second) => first > second;
159
160    /// <summary>

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLinkAddress = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 internal links size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16     public unsafe abstract class UInt64InternalLinksSizeBalancedTreeMethodsBase :
        ↪ 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
71     /// <summary>
72     /// <para>
73     ///     Gets the zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <returns>
78     /// <para>The ulong</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override ulong GetZero() => OUL;
83
84     /// <summary>
85     /// <para>
86     ///     Determines whether this instance equal to zero.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// </param>
93     /// <returns>
94     /// <para>The bool</para>
95     /// <para></para>
96     /// </returns>
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     protected override bool EqualToZero(ulong value) => value == OUL;
99
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     /// </param>
110     /// <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>
416    /// <para></para>
417    /// </param>
418    /// <param name="second">
419    /// <para>The second.</para>
420    /// <para></para>
421    /// </param>
422    /// <returns>
423    /// <para>The bool</para>
424    /// <para></para>
425    /// </returns>

```

```

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

```

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

```

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

```



```

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>
79     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref _linksIndexParts[link];
80 }

```

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

```

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

```

```

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 using the specified node.
184    /// </para>
185    /// <para></para>
186    /// </summary>
187    /// <param name="node">
188    /// <para>The node.</para>
189    /// <para></para>
190    /// </param>
191    /// <returns>
192    /// <para>The link</para>
193    /// <para></para>
194    /// </returns>
195    [MethodImpl(MethodImplOptions.AggressiveInlining)]
196    protected override TLinkAddress GetTreeRoot(TLinkAddress node) =>
197        ↪ LinksIndexParts[node].RootAsSource;
198
199    /// <summary>
200    /// <para>
201    /// Gets the base part value using the specified node.
202    /// </para>
203    /// <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 }

```

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

```

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

```

```

5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 internal links sources size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt64InternalLinksSourcesSizeBalancedTreeMethods :
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>
79        [MethodImpl(MethodImplOptions.AggressiveInlining)]
80        protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
81        ↪ LinksIndexParts[node].RightAsSource;
82    }
83 }

```

```

76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLinkAddress GetLeft(TLinkAddress node) =>
92         ↪ LinksIndexParts[node].LeftAsSource;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLinkAddress GetRight(TLinkAddress node) =>
110        ↪ LinksIndexParts[node].RightAsSource;
111
112    /// <summary>
113    /// <para>
114    /// Sets the left using the specified node.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="node">
119    /// <para>The node.</para>
120    /// <para></para>
121    /// </param>
122    /// <param name="left">
123    /// <para>The left.</para>
124    /// <para></para>
125    /// </param>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
128        ↪ LinksIndexParts[node].LeftAsSource = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">
141    /// <para>The right.</para>
142    /// <para></para>
143    /// </param>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
146        ↪ LinksIndexParts[node].RightAsSource = right;
147
148    /// <summary>
149    /// <para>
150    /// Gets the size using the specified node.
151    /// </para>
152    /// <para></para>
153    /// </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>
222    /// <param name="node">
223    /// <para>The node.</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     /// </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) =>
267         ↪ SearchCore(GetTreeRoot(source), target);
268 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16         ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">

```



```

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">
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].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     {
246         ref var link = ref LinksIndexParts[node];
247         link.LeftAsTarget = Zero;
248         link.RightAsTarget = Zero;
249         link.SizeAsTarget = Zero;
250     }

```

```

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 }

```

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

```

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

160
161 /// <summary>
162 /// <para>
163 /// Sets the size using the specified node.
164 /// </para>
165 /// <para></para>
166 /// </summary>
167 /// <param name="node">
168 /// <para>The node.</para>
169 /// <para></para>
170 /// </param>
171 /// <param name="size">
172 /// <para>The size.</para>
173 /// <para></para>
174 /// </param>
175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
176 protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
    ↳ LinksIndexParts[node].SizeAsTarget = size;

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

194
195 /// <summary>

```

```

196     /// <para>
197     /// Gets the base part value using the specified node.
198     /// </para>
199     /// <para></para>
200     /// </summary>
201     /// <param name="node">
202     /// <para>The node.</para>
203     /// <para></para>
204     /// </param>
205     /// <returns>
206     /// <para>The link</para>
207     /// <para></para>
208     /// </returns>
209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
210     protected override 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);
270 }

```

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

```

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

```



```

75     /// <para></para>
76     /// </param>
77     /// <param name="indexMemory">
78     /// <para>A index memory.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="memoryReservationStep">
82     /// <para>A memory reservation step.</para>
83     /// <para></para>
84     /// </param>
85     /// <param name="constants">
86     /// <para>A constants.</para>
87     /// <para></para>
88     /// </param>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants) :
        ↪ this(dataMemory, indexMemory, memoryReservationStep, constants,
        ↪ IndexTreeType.Default, useLinkedList: true) { }

91
92     /// <summary>
93     /// <para>
94     /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <param name="dataMemory">
99     /// <para>A data memory.</para>
100    /// <para></para>
101    /// </param>
102    /// <param name="indexMemory">
103    /// <para>A index memory.</para>
104    /// <para></para>
105    /// </param>
106    /// <param name="memoryReservationStep">
107    /// <para>A memory reservation step.</para>
108    /// <para></para>
109    /// </param>
110    /// <param name="constants">
111    /// <para>A constants.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="indexTreeType">
115    /// <para>A index tree type.</para>
116    /// <para></para>
117    /// </param>
118    /// <param name="useLinkedList">
119    /// <para>A use linked list.</para>
120    /// <para></para>
121    /// </param>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants,
        ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↪ memoryReservationStep, constants, useLinkedList)
124    {
125        if (indexTreeType == IndexTreeType.SizeBalancedTree)
126        {
127            _createInternalSourceTreeMethods = () => new
                ↪ UInt64InternalLinksSourcesSizeBalancedTreeMethods(Constants,
                ↪ _linksDataParts, _linksIndexParts, _header);
128            _createExternalSourceTreeMethods = () => new
                ↪ UInt64ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
                ↪ _linksDataParts, _linksIndexParts, _header);
129            _createInternalTargetTreeMethods = () => new
                ↪ UInt64InternalLinksTargetsSizeBalancedTreeMethods(Constants,
                ↪ _linksDataParts, _linksIndexParts, _header);
130            _createExternalTargetTreeMethods = () => new
                ↪ UInt64ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
                ↪ _linksDataParts, _linksIndexParts, _header);
131        }
132        else
133        {
134            _createInternalSourceTreeMethods = () => new
                ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
                ↪ _linksDataParts, _linksIndexParts, _header);

```

```

135         _createExternalSourceTreeMethods = () => new
136         ↪ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
137         ↪ _linksDataParts, _linksIndexParts, _header);
138     _createInternalTargetTreeMethods = () => new
139     ↪ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
140     ↪ _linksDataParts, _linksIndexParts, _header);
141     _createExternalTargetTreeMethods = () => new
142     ↪ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
143     ↪ _linksDataParts, _linksIndexParts, _header);
144 }
145 Init(dataMemory, indexMemory);
146 }
147
148 /// <summary>
149 /// <para>
150 /// Sets the pointers using the specified data memory.
151 /// </para>
152 /// <para></para>
153 /// </summary>
154 /// <param name="dataMemory">
155 /// <para>The data memory.</para>
156 /// <para></para>
157 /// </param>
158 /// <param name="indexMemory">
159 /// <para>The index memory.</para>
160 /// <para></para>
161 /// </param>
162 [MethodImpl(MethodImplOptions.AggressiveInlining)]
163 protected override void SetPointers(IResizableDirectMemory dataMemory,
164 ↪ IResizableDirectMemory indexMemory)
165 {
166     _linksDataParts = (RawLinkDataPart<TLinkAddress>*)dataMemory.Pointer;
167     _linksIndexParts = (RawLinkIndexPart<TLinkAddress>*)indexMemory.Pointer;
168     _header = (LinksHeader<TLinkAddress>*)indexMemory.Pointer;
169     if (_useLinkedList)
170     {
171         InternalSourcesListMethods = new
172         ↪ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
173         ↪ _linksIndexParts);
174     }
175     else
176     {
177         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
178     }
179     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
180     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
181     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
182     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_linksDataParts, _header);
183 }
184
185 /// <summary>
186 /// <para>
187 /// Resets the pointers.
188 /// </para>
189 /// <para></para>
190 /// </summary>
191 [MethodImpl(MethodImplOptions.AggressiveInlining)]
192 protected override void ResetPointers()
193 {
194     base.ResetPointers();
195     _linksDataParts = null;
196     _linksIndexParts = null;
197     _header = null;
198 }
199
200 /// <summary>
201 /// <para>
202 /// Gets the header reference.
203 /// </para>
204 /// <para></para>
205 /// </summary>
206 /// <returns>
207 /// <para>A ref links header of t link</para>
208 /// <para></para>
209 /// </returns>
210 [MethodImpl(MethodImplOptions.AggressiveInlining)]
211 protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref *_header;

```

```

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>
279    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override bool LessThan(ulong first, ulong second) => first < second;

```

```

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() => OUL;
355
356     /// <summary>
357     /// <para>

```

```

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

```

```

436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The ulong</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override ulong Subtract(ulong first, ulong second) => first - second;
444
445     /// <summary>
446     /// <para>
447     /// Increments the link.
448     /// </para>
449     /// <para></para>
450     /// </summary>
451     /// <param name="link">
452     /// <para>The link.</para>
453     /// <para></para>
454     /// </param>
455     /// <returns>
456     /// <para>The ulong</para>
457     /// <para></para>
458     /// </returns>
459     [MethodImpl(MethodImplOptions.AggressiveInlining)]
460     protected override ulong Increment(ulong link) => ++link;
461
462     /// <summary>
463     /// <para>
464     /// Decrements the link.
465     /// </para>
466     /// <para></para>
467     /// </summary>
468     /// <param name="link">
469     /// <para>The link.</para>
470     /// <para></para>
471     /// </param>
472     /// <returns>
473     /// <para>The ulong</para>
474     /// <para></para>
475     /// </returns>
476     [MethodImpl(MethodImplOptions.AggressiveInlining)]
477     protected override ulong Decrement(ulong link) => --link;
478 }
479 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using 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 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using Platform.Numbers;
9  using static System.Runtime.CompilerServices.Unsafe;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.Memory.United.Generic
14 {
15     /// <summary>
16     /// <para>
17     /// Represents the links avl balanced tree methods base.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     /// <seealso cref="SizedAndThreadedAVLBalancedTreeMethods{TLinkAddress}"/>
22     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
23     public unsafe abstract class LinksAvlBalancedTreeMethodsBase<TLinkAddress> :
    ↪ 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*
80     ↪ links, byte* header)
81     {
82         Links = links;
83         Header = header;
84         Break = constants.Break;
85         Continue = constants.Continue;
86     }
87     /// <summary>
88     /// <para>
89     /// Gets the tree root.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <returns>
94     /// <para>The link</para>
95     /// <para></para>
96     /// </returns>
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     protected abstract TLinkAddress GetTreeRoot();
99
100    /// <summary>
101    /// <para>
102    /// Gets the base part value using the specified link.
103    /// </para>
104    /// <para></para>
105    /// </summary>
106    /// <param name="link">
107    /// <para>The link.</para>

```



```

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

```

```

184     /// </returns>
185     [MethodImpl(MethodImplOptions.AggressiveInlining)]
186     protected virtual ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
        ↳ AsRef<LinksHeader<TLinkAddress>>(Header);
187
188     /// <summary>
189     /// <para>
190     /// Gets the link reference using the specified link.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     /// <param name="link">
195     /// <para>The link.</para>
196     /// <para></para>
197     /// </param>
198     /// <returns>
199     /// <para>A ref raw link of t link</para>
200     /// <para></para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     protected virtual ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
        ↳ AsRef<RawLink<TLinkAddress>>(Links + (RawLink<TLinkAddress>.SizeInBytes *
        ↳ _addressToInt64Converter.Convert(link)));
204
205     /// <summary>
206     /// <para>
207     /// Gets the link values using the specified link index.
208     /// </para>
209     /// <para></para>
210     /// </summary>
211     /// <param name="linkIndex">
212     /// <para>The link index.</para>
213     /// <para></para>
214     /// </param>
215     /// <returns>
216     /// <para>A list of t link</para>
217     /// <para></para>
218     /// </returns>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
221     {
222         ref var link = ref GetLinkReference(linkIndex);
223         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
224     }
225
226     /// <summary>
227     /// <para>
228     /// Determines whether this instance first is to the left of second.
229     /// </para>
230     /// <para></para>
231     /// </summary>
232     /// <param name="first">
233     /// <para>The first.</para>
234     /// <para></para>
235     /// </param>
236     /// <param name="second">
237     /// <para>The second.</para>
238     /// <para></para>
239     /// </param>
240     /// <returns>
241     /// <para>The bool</para>
242     /// <para></para>
243     /// </returns>
244     [MethodImpl(MethodImplOptions.AggressiveInlining)]
245     protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
246     {
247         ref var firstLink = ref GetLinkReference(first);
248         ref var secondLink = ref GetLinkReference(second);
249         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
        ↳ secondLink.Source, secondLink.Target);
250     }
251
252     /// <summary>
253     /// <para>
254     /// Determines whether this instance first is to the right of second.
255     /// </para>
256     /// <para></para>
257     /// </summary>

```

```

258     /// <param name="first">
259     /// <para>The first.</para>
260     /// <para></para>
261     /// </param>
262     /// <param name="second">
263     /// <para>The second.</para>
264     /// <para></para>
265     /// </param>
266     /// <returns>
267     /// <para>The bool</para>
268     /// <para></para>
269     /// </returns>
270     [MethodImpl(MethodImplOptions.AggressiveInlining)]
271     protected override bool FirstIsToTheRightOfSecond(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 /// </param>
347 /// <param name="value">
348 /// <para>The value.</para>
349 /// </param>
350 [MethodImpl(MethodImplOptions.AggressiveInlining)]
351 protected virtual void SetLeftIsChildValue(ref TLinkAddress storedValue, bool value)
352 {
353     unchecked
354     {
355         var previousValue = storedValue;
356         var modified = Bit<TLinkAddress>.PartialWrite(previousValue,
357             ↪ _boolToAddressConverter.Convert(value), 4, 1);
358         storedValue = modified;
359     }
360 }
361
362 /// <summary>
363 /// <para>
364 /// Determines whether this instance get right is child value.
365 /// </para>
366 /// <para></para>
367 /// </summary>
368 /// <param name="value">
369 /// <para>The value.</para>
370 /// </param>
371 /// <returns>
372 /// <para>The bool</para>
373 /// <para></para>
374 /// </returns>
375 [MethodImpl(MethodImplOptions.AggressiveInlining)]
376 protected virtual bool GetRightIsChildValue(TLinkAddress value)
377 {
378     unchecked
379     {
380         return _addressToBoolConverter.Convert(Bit<TLinkAddress>.PartialRead(value, 3,
381             ↪ 1));
382         //return !EqualityComparer.Equals(Bit<TLinkAddress>.PartialRead(value, 3, 1),
383             ↪ default);
384     }
385 }
386
387 /// <summary>
388 /// <para>
389 /// Sets the right is child value using the specified stored value.
390 /// </para>
391 /// <para></para>
392 /// </summary>
393 /// <param name="storedValue">
394 /// <para>The stored value.</para>
395 /// </param>
396 /// <param name="value">
397 /// <para>The value.</para>
398 /// </param>
399 [MethodImpl(MethodImplOptions.AggressiveInlining)]
400 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 /// <summary>
411 /// <para>
412 /// Determines whether this instance is child.
413 /// </para>
414 /// <para></para>
415 /// </summary>
416 /// <param name="parent">
417 /// <para>The parent.</para>
418 /// <para></para>
419 /// </param>
420 /// <param name="possibleChild">
421 /// <para>The possible child.</para>
422 /// <para></para>
423 /// </param>
424 /// <returns>
425 /// <para>The bool</para>
426 /// <para></para>
427 /// </returns>
428 [MethodImpl(MethodImplOptions.AggressiveInlining)]
429 protected bool IsChild(TLinkAddress parent, TLinkAddress possibleChild)
430 {
431     var parentSize = GetSize(parent);
432     var childSize = GetSizeOrZero(possibleChild);
433     return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
434 }
435
436 /// <summary>
437 /// <para>
438 /// Gets the balance value using the specified stored value.
439 /// </para>
440 /// <para></para>
441 /// </summary>
442 /// <param name="storedValue">
443 /// <para>The stored value.</para>
444 /// <para></para>
445 /// </param>
446 /// <returns>
447 /// <para>The sbyte</para>
448 /// <para></para>
449 /// </returns>
450 [MethodImpl(MethodImplOptions.AggressiveInlining)]
451 protected virtual sbyte GetBalanceValue(TLinkAddress storedValue)
452 {
453     unchecked
454     {
455         var value =
456             ↪ _addressToInt32Converter.Convert(Bit<TLinkAddress>.PartialRead(storedValue,
457                 ↪ 0, 3));
458         value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
459             ↪ end of sbyte
460         return (sbyte)value;
461     }
462 }
463
464 /// <summary>
465 /// <para>
466 /// Sets the balance value using the specified stored value.
467 /// </para>
468 /// <para></para>
469 /// </summary>
470 /// <param name="storedValue">
471 /// <para>The stored value.</para>
472 /// <para></para>
473 /// </param>
474 /// <param name="value">
475 /// <para>The value.</para>
476 /// <para></para>
477 /// </param>
478 [MethodImpl(MethodImplOptions.AggressiveInlining)]
479 protected virtual void SetBalanceValue(ref TLinkAddress storedValue, sbyte value)
480 {

```

```

478 unchecked
479 {
480     var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
        ↪ value & 3);
481     var modified = Bit<TLinkAddress>.PartialWrite(storedValue, packagedValue, 0, 3);
482     storedValue = modified;
483 }
484 }
485
486 /// <summary>
487 /// <para>
488 /// The zero.
489 /// </para>
490 /// <para></para>
491 /// </summary>
492 public TLinkAddress this[TLinkAddress index]
493 {
494     [MethodImpl(MethodImplOptions.AggressiveInlining)]
495     get
496     {
497         var root = GetTreeRoot();
498         if (GreaterOrEqualThan(index, GetSize(root)))
499         {
500             return Zero;
501         }
502         while (!EqualToZero(root))
503         {
504             var left = GetLeftOrDefault(root);
505             var leftSize = GetSizeOrZero(left);
506             if (LessThan(index, leftSize))
507             {
508                 root = left;
509                 continue;
510             }
511             if (AreEqual(index, leftSize))
512             {
513                 return root;
514             }
515             root = GetRightOrDefault(root);
516             index = Subtract(index, Increment(leftSize));
517         }
518         return Zero; // TODO: Impossible situation exception (only if tree structure
        ↪ broken)
519     }
520 }
521
522 /// <summary>
523 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
524 /// </summary>
525 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
526 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
527 /// <returns>Индекс искомой связи.</returns>
528 [MethodImpl(MethodImplOptions.AggressiveInlining)]
529 public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
530 {
531     var root = GetTreeRoot();
532     while (!EqualToZero(root))
533     {
534         ref var rootLink = ref GetLinkReference(root);
535         var rootSource = rootLink.Source;
536         var rootTarget = rootLink.Target;
537         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
            ↪ node.Key < root.Key
538         {
539             root = GetLeftOrDefault(root);
540         }
541         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
            ↪ node.Key > root.Key
542         {
543             root = GetRightOrDefault(root);
544         }
545         else // node.Key == root.Key
546         {
547             return root;
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     var root = GetTreeRoot();
627     if (EqualToZero(root))
628 
```

```

629     {
630         return Continue;
631     }
632     TLinkAddress first = Zero, current = root;
633     while (!EqualToZero(current))
634     {
635         var @base = GetBasePartValue(current);
636         if (GreaterOrEqualThan(@base, link))
637         {
638             if (AreEqual(@base, link))
639             {
640                 first = current;
641             }
642             current = GetLeftOrDefault(current);
643         }
644         else
645         {
646             current = GetRightOrDefault(current);
647         }
648     }
649     if (!EqualToZero(first))
650     {
651         current = first;
652         while (true)
653         {
654             if (AreEqual(handler(GetLinkValues(current)), Break))
655             {
656                 return Break;
657             }
658             current = GetNext(current);
659             if (EqualToZero(current) || !AreEqual(GetBasePartValue(current), link))
660             {
661                 break;
662             }
663         }
664     }
665     return Continue;
666 }
667
668 /// <summary>
669 /// <para>
670 /// Prints the node value using the specified node.
671 /// </para>
672 /// <para></para>
673 /// </summary>
674 /// <param name="node">
675 /// <para>The node.</para>
676 /// <para></para>
677 /// </param>
678 /// <param name="sb">
679 /// <para>The sb.</para>
680 /// <para></para>
681 /// </param>
682 [MethodImpl(MethodImplOptions.AggressiveInlining)]
683 protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
684 {
685     ref var link = ref GetLinkReference(node);
686     sb.Append(' ');
687     sb.Append(link.Source);
688     sb.Append('-');
689     sb.Append('>');
690     sb.Append(link.Target);
691 }
692 }
693 }

```

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

```

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

```



```

13 {
14     /// <summary>
15     /// <para>
16     /// Represents the links recursionless size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLinkAddress}"/>
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class LinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress> :
    ↳ 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.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         protected readonly byte* Links;
47         /// <summary>
48         /// <para>
49         /// The header.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         protected readonly byte* Header;
54
55         /// <summary>
56         /// <para>
57         /// Initializes a new <see cref="LinksRecursionlessSizeBalancedTreeMethodsBase"/>
    ↳ instance.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         /// <param name="constants">
62         /// <para>A constants.</para>
63         /// <para></para>
64         /// </param>
65         /// <param name="links">
66         /// <para>A links.</para>
67         /// <para></para>
68         /// </param>
69         /// <param name="header">
70         /// <para>A header.</para>
71         /// <para></para>
72         /// </param>
73         [MethodImpl(MethodImplOptions.AggressiveInlining)]
74         protected LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
    ↳ constants, byte* links, byte* header)
75         {
76             Links = links;
77             Header = header;
78             Break = constants.Break;
79             Continue = constants.Continue;
80         }
81
82         /// <summary>
83         /// <para>
84         /// Gets the tree root.
85         /// </para>
86         /// <para></para>
87         /// </summary>

```

```

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

```

```

165 /// <para></para>
166 /// </returns>
167 [MethodImpl(MethodImplOptions.AggressiveInlining)]
168 protected abstract bool FirstIsToTheLeftOfSecond(TLinkAddress source, TLinkAddress
169     ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);
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
182     ↪ AsRef<LinksHeader<TLinkAddress>>(Header);
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
199     ↪ AsRef<RawLink<TLinkAddress>>(Links + (RawLink<TLinkAddress>.SizeInBytes *
200     ↪ _addressToInt64Converter.Convert(link)));
201 /// <summary>
202 /// <para>
203 /// Gets the link values using the specified link index.
204 /// </para>
205 /// <para></para>
206 /// </summary>
207 /// <param name="linkIndex">
208 /// <para>The link index.</para>
209 /// <para></para>
210 /// </param>
211 /// <returns>
212 /// <para>A list of t link</para>
213 /// <para></para>
214 /// </returns>
215 [MethodImpl(MethodImplOptions.AggressiveInlining)]
216 protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
217 {
218     ref var link = ref GetLinkReference(linkIndex);
219     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
220 }
221 /// <summary>
222 /// <para>
223 /// Determines whether this instance first is to the left of second.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="first">
228 /// <para>The first.</para>
229 /// <para></para>
230 /// </param>
231 /// <param name="second">
232 /// <para>The second.</para>
233 /// <para></para>
234 /// </param>
235 /// <returns>
236 /// <para>The bool</para>
237 /// <para></para>
238 /// </returns>

```

```

239 [MethodImpl(MethodImplOptions.AggressiveInlining)]
240 protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
241 {
242     ref var firstLink = ref GetLinkReference(first);
243     ref var secondLink = ref GetLinkReference(second);
244     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
245 }
246
247 /// <summary>
248 /// <para>
249 /// Determines whether this instance first is to the right of second.
250 /// </para>
251 /// <para></para>
252 /// </summary>
253 /// <param name="first">
254 /// <para>The first.</para>
255 /// <para></para>
256 /// </param>
257 /// <param name="second">
258 /// <para>The second.</para>
259 /// <para></para>
260 /// </param>
261 /// <returns>
262 /// <para>The bool</para>
263 /// <para></para>
264 /// </returns>
265 [MethodImpl(MethodImplOptions.AggressiveInlining)]
266 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
    ↪ second)
267 {
268     ref var firstLink = ref GetLinkReference(first);
269     ref var secondLink = ref GetLinkReference(second);
270     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
271 }
272
273 /// <summary>
274 /// <para>
275 /// The zero.
276 /// </para>
277 /// <para></para>
278 /// </summary>
279 public TLinkAddress this[TLinkAddress index]
280 {
281     [MethodImpl(MethodImplOptions.AggressiveInlining)]
282     get
283     {
284         var root = GetTreeRoot();
285         if (GreaterOrEqualThan(index, GetSize(root)))
286         {
287             return Zero;
288         }
289         while (!EqualToZero(root))
290         {
291             var left = GetLeftOrDefault(root);
292             var leftSize = GetSizeOrZero(left);
293             if (LessThan(index, leftSize))
294             {
295                 root = left;
296                 continue;
297             }
298             if (AreEqual(index, leftSize))
299             {
300                 return root;
301             }
302             root = GetRightOrDefault(root);
303             index = Subtract(index, Increment(leftSize));
304         }
305         return Zero; // TODO: Impossible situation exception (only if tree structure
            ↪ broken)
306     }
307 }
308
309 /// <summary>
310 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
311 /// </summary>

```

```

312 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
313 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
314 /// <returns>Индекс искомой связи.</returns>
315 [MethodImpl(MethodImplOptions.AggressiveInlining)]
316 public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
317 {
318     var root = GetTreeRoot();
319     while (!EqualToZero(root))
320     {
321         ref var rootLink = ref GetLinkReference(root);
322         var rootSource = rootLink.Source;
323         var rootTarget = rootLink.Target;
324         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
325             ↪ node.Key < root.Key
326         {
327             root = GetLeftOrDefault(root);
328         }
329         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
330             ↪ node.Key > root.Key
331         {
332             root = GetRightOrDefault(root);
333         }
334         else // node.Key == root.Key
335         {
336             return root;
337         }
338     }
339     return Zero;
340 }
341
342 // TODO: Return indices range instead of references count
343 /// <summary>
344 /// <para>
345 /// Counts the usages using the specified link.
346 /// </para>
347 /// <para></para>
348 /// </summary>
349 /// <param name="link">
350 /// <para>The link.</para>
351 /// <para></para>
352 /// </param>
353 /// <returns>
354 /// <para>The link</para>
355 /// <para></para>
356 /// </returns>
357 [MethodImpl(MethodImplOptions.AggressiveInlining)]
358 public TLinkAddress CountUsages(TLinkAddress link)
359 {
360     var root = GetTreeRoot();
361     var total = GetSize(root);
362     var totalRightIgnore = Zero;
363     while (!EqualToZero(root))
364     {
365         var @base = GetBasePartValue(root);
366         if (LessOrEqualThan(@base, link))
367         {
368             root = GetRightOrDefault(root);
369         }
370         else
371         {
372             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
373             root = GetLeftOrDefault(root);
374         }
375     }
376     root = GetTreeRoot();
377     var totalLeftIgnore = Zero;
378     while (!EqualToZero(root))
379     {
380         var @base = GetBasePartValue(root);
381         if (GreaterOrEqualThan(@base, link))
382         {
383             root = GetLeftOrDefault(root);
384         }
385         else
386         {
387             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
388             root = GetRightOrDefault(root);
389         }
390     }
391 }

```

```

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 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.United.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the links size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizeBalancedTreeMethods{TLinkAddress}"/>
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class LinksSizeBalancedTreeMethodsBase<TLinkAddress> :
23         ↳ SizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
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         ///
63         /// </para>
64         /// </summary>
65     }
66 }

```

```

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

```



```

134    /// <returns>
135    /// <para>The bool</para>
136    /// <para></para>
137    /// </returns>
138    [MethodImpl(MethodImplOptions.AggressiveInlining)]
139    protected abstract bool FirstIsToTheRightOfSecond(TLinkAddress source, TLinkAddress
    ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);

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

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

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

199
200    /// <summary>
201    /// <para>
202    /// Gets the link values using the specified link index.
203    /// </para>
204    /// <para></para>
205    /// </summary>

```

```

206 /// <param name="linkIndex">
207 /// <para>The link index.</para>
208 /// <para></para>
209 /// </param>
210 /// <returns>
211 /// <para>A list of t link</para>
212 /// <para></para>
213 /// </returns>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
216 {
217     ref var link = ref GetLinkReference(linkIndex);
218     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
219 }
220
221 /// <summary>
222 /// <para>
223 /// Determines whether this instance first is to the left of second.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="first">
228 /// <para>The first.</para>
229 /// <para></para>
230 /// </param>
231 /// <param name="second">
232 /// <para>The second.</para>
233 /// <para></para>
234 /// </param>
235 /// <returns>
236 /// <para>The bool</para>
237 /// <para></para>
238 /// </returns>
239 [MethodImpl(MethodImplOptions.AggressiveInlining)]
240 protected override bool 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>
281 /// </summary>
282 public TLinkAddress this[TLinkAddress index]
283 {

```

```

281 [MethodImpl(MethodImplOptions.AggressiveInlining)]
282 get
283 {
284     var root = GetTreeRoot();
285     if (GreaterOrEqualThan(index, GetSize(root)))
286     {
287         return Zero;
288     }
289     while (!EqualToZero(root))
290     {
291         var left = GetLeftOrDefault(root);
292         var leftSize = GetSizeOrZero(left);
293         if (LessThan(index, leftSize))
294         {
295             root = left;
296             continue;
297         }
298         if (AreEqual(index, leftSize))
299         {
300             return root;
301         }
302         root = GetRightOrDefault(root);
303         index = Subtract(index, Increment(leftSize));
304     }
305     return Zero; // TODO: Impossible situation exception (only if tree structure
306                 ↪ broken)
307 }
308
309 /// <summary>
310 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
311 ↪ (концом).
312 /// </summary>
313 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
314 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
315 /// <returns>Индекс искомой связи.</returns>
316 [MethodImpl(MethodImplOptions.AggressiveInlining)]
317 public 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>

```

```

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))
430         {
431             return @break;
432         }
433     }
434 }

```

```

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

```

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

```

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

```

```

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>
174    [MethodImpl(MethodImplOptions.AggressiveInlining)]
175    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
176        ↪ SetSizeValue(ref GetLinkReference(node).SizeAsSource, size);

```

```

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(TLinkAddress node) =>
188     ↪ 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) =>
206     ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance get right is child.
211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <param name="node">
215     /// <para>The node.</para>
216     /// <para></para>
217     /// </param>
218     /// <returns>
219     /// <para>The bool</para>
220     /// <para></para>
221     /// </returns>
222     [MethodImpl(MethodImplOptions.AggressiveInlining)]
223     protected override bool GetRightIsChild(TLinkAddress node) =>
224     ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsSource);
225
226     /// <summary>
227     /// <para>
228     /// Sets the right is child using the specified node.
229     /// </para>
230     /// <para></para>
231     /// </summary>
232     /// <param name="node">
233     /// <para>The node.</para>
234     /// <para></para>
235     /// </param>
236     /// <param name="value">
237     /// <para>The value.</para>
238     /// <para></para>
239     /// </param>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     protected override void SetRightIsChild(TLinkAddress node, bool value) =>
242     ↪ SetRightIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
243
244     /// <summary>
245     /// <para>
246     /// Gets the balance using the specified node.
247     /// </para>
248     /// <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>
320    /// <para></para>
321    /// </param>
322    /// <param name="secondSource">
323    /// <para>The second source.</para>

```

```

321     /// <para></para>
322     /// </param>
323     /// <param name="secondTarget">
324     /// <para>The second target.</para>
325     /// <para></para>
326     /// </param>
327     /// <returns>
328     /// <para>The bool</para>
329     /// <para></para>
330     /// </returns>
331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
332     protected override bool FirstIsToTheLeftOfSecond(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 }

```

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

```

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         /// <para></para>
27         /// </param>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// <para></para>
35         /// </param>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
38             ↳ constants, byte* links, byte* header) : base(constants, links, header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="node">
47         /// <para>The node.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The ref link</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
56             ↳ GetLinkReference(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
74             ↳ GetLinkReference(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>

```

```

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>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override TLinkAddress GetSize(TLinkAddress node) =>
159        ↪ GetLinkReference(node).SizeAsSource;

```

```

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

```

```

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 }

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links sources size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="LinksSizeBalancedTreeMethodsBase{TLinkAddress}"/>
14    public unsafe class LinksSourcesSizeBalancedTreeMethods<TLinkAddress> :
        ↪ 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>
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).SizeAsSource = 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().RootAsSource;
185
186     /// <summary>
187     /// <para>
188     /// Gets the base part value using the specified link.
189     /// </para>
190     /// <para></para>
191     /// </summary>
192     /// <param name="link">
193     /// <para>The link.</para>
194     /// <para></para>
195     /// </param>
196     /// <returns>
197     /// <para>The link</para>
198     /// <para></para>
199     /// </returns>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
202         ↪ GetLinkReference(link).Source;
203
204     /// <summary>
205     /// <para>
206     /// Determines whether this instance first is to the left of second.
207     /// </para>
208     /// <para></para>
209     /// </summary>
210     /// <param name="firstSource">
211     /// <para>The first source.</para>
212     /// <para></para>
213     /// </param>
214     /// <param name="firstTarget">
215     /// <para>The first target.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="secondSource">
219     /// <para>The second source.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondTarget">
223     /// <para>The second target.</para>
224     /// <para></para>
225     /// </param>
226     /// <returns>
227     /// <para>The bool</para>
228     /// <para></para>
229     /// </returns>
230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
231     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
232         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
233         ↪ LessThan(firstSource, secondSource) || (AreEqual(firstSource, secondSource) &&
234         ↪ LessThan(firstTarget, secondTarget));
235
236     /// <summary>
237     /// <para>
238     /// Determines whether this instance first is to the right of second.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="firstSource">

```

```

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

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

```

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

```

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

```

```

31    /// <para>A header.</para>
32    /// <para></para>
33    /// </param>
34    [MethodImpl(MethodImplOptions.AggressiveInlining)]
35    public LinksTargetsAvlBalancedTreeMethods(LinksConstants<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) =>
121         ↪ GetLinkReference(node).LeftAsTarget = left;
122
123     /// <summary>
124     /// <para>
125     /// Sets the right using the specified node.
126     /// </para>
127     /// <para></para>
128     /// </summary>
129     /// <param name="node">
130     /// <para>The node.</para>
131     /// <para></para>
132     /// </param>
133     /// <param name="right">
134     /// <para>The right.</para>
135     /// <para></para>
136     /// </param>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
139         ↪ GetLinkReference(node).RightAsTarget = right;
140
141     /// <summary>
142     /// <para>
143     /// Gets the size using the specified node.
144     /// </para>
145     /// <para></para>
146     /// </summary>
147     /// <param name="node">
148     /// <para>The node.</para>
149     /// <para></para>
150     /// </param>
151     /// <returns>
152     /// <para>The link</para>
153     /// <para></para>
154     /// </returns>
155     [MethodImpl(MethodImplOptions.AggressiveInlining)]
156     protected override TLinkAddress GetSize(TLinkAddress node) =>
157         ↪ GetSizeValue(GetLinkReference(node).SizeAsTarget);
158
159     /// <summary>
160     /// <para>
161     /// Sets the size using the specified node.
162     /// </para>
163     /// <para></para>
164     /// </summary>
165     /// <param name="node">
166     /// <para>The node.</para>
167     /// <para></para>
168     /// </param>
169     /// <param name="size">
170     /// <para>The size.</para>
171     /// <para></para>
172     /// </param>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
175         ↪ SetSizeValue(ref GetLinkReference(node).SizeAsTarget, size);
176
177     /// <summary>
178     /// <para>
179     /// Determines whether this instance get left is child.
180     /// </para>
181     /// <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     /// <para>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     /// <para>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     /// <para>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     /// <para>Gets the balance using the specified node.
248     /// </para>
249     /// <para></para>
250     /// </summary>
251     /// <param name="node">
252     /// <para>The node.</para>
253     /// <para></para>
254     /// </param>
255     /// <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     /// </summary>
264     /// <param name="node">
265     /// <para>The node.</para>
266     /// <para></para>
267     /// </param>
268     /// <param name="value">
269     /// <para>The value.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void SetBalance(TLinkAddress node, sbyte value) =>
274         ↪ SetBalanceValue(ref GetLinkReference(node).SizeAsTarget, value);
275
276     /// <summary>
277     /// <para>
278     /// Gets the tree root.
279     /// </para>
280     /// <para></para>
281     /// </summary>
282     /// <returns>
283     /// <para>The link</para>
284     /// <para></para>
285     /// </returns>
286     [MethodImpl(MethodImplOptions.AggressiveInlining)]
287     protected override TLinkAddress GetTreeRoot() => GetHeaderReference().RootAsTarget;
288
289     /// <summary>
290     /// <para>
291     /// Gets the base part value using the specified link.
292     /// </para>
293     /// <para></para>
294     /// </summary>
295     /// <param name="link">
296     /// <para>The link.</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 GetBasePartValue(TLinkAddress link) =>
305         ↪ GetLinkReference(link).Target;
306
307     /// <summary>
308     /// <para>
309     /// Determines whether this instance first is to the left of second.
310     /// </para>
311     /// <para></para>
312     /// </summary>
313     /// <param name="firstSource">
314     /// <para>The first source.</para>
315     /// <para></para>
316     /// </param>
317     /// <param name="firstTarget">
318     /// <para>The first target.</para>
319     /// <para></para>
320     /// </param>
321     /// <param name="secondSource">
322     /// <para>The second source.</para>
323     /// <para></para>
324     /// </param>
325     /// <param name="secondTarget">
326     /// <para>The second target.</para>
327     /// <para></para>
328     /// </param>

```

```

327     /// <returns>
328     /// <para>The bool</para>
329     /// <para></para>
330     /// </returns>
331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
332     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
        ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
        ↪ LessThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
        ↪ LessThan(firstSource, secondSource));
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(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
        ↪ GreaterThan(firstSource, secondSource));
362
363     /// <summary>
364     /// <para>
365     /// Clears the node using the specified node.
366     /// </para>
367     /// <para></para>
368     /// </summary>
369     /// <param name="node">
370     /// <para>The node.</para>
371     /// <para></para>
372     /// </param>
373     [MethodImpl(MethodImplOptions.AggressiveInlining)]
374     protected override void ClearNode(TLinkAddress node)
375     {
376         ref var link = ref GetLinkReference(node);
377         link.LeftAsTarget = Zero;
378         link.RightAsTarget = Zero;
379         link.SizeAsTarget = Zero;
380     }
381 }
382 }

```

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

```

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"/>
    ↳ 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 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;
160
161    /// <summary>
162    /// <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)]
232     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
233         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
234         ↪ LessThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
235         ↪ LessThan(firstSource, secondSource));

```

```

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

```

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

```

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

```

```

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">
171    /// <para>The size.</para>
172    /// <para></para>
173    /// </param>
174    [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)]
232 protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
233     ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
234     ↳ LessThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
235     ↳ LessThan(firstSource, secondSource));
236
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">

```

```

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     /// <summary>
261     /// <para>
262     /// 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.LeftAsTarget = Zero;
275         link.RightAsTarget = Zero;
276         link.SizeAsTarget = Zero;
277     }
278 }
279 }
280 }

```

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

```

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

```

```

37
38 /// <summary>
39 /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
40   ↳ минимальным шагом расширения базы данных.
41 /// </summary>
42 /// <param name="address">Полный путь к файлу базы данных.</param>
43 /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
44   ↳ байтах.</param>
45 [MethodImpl(MethodImplOptions.AggressiveInlining)]
46 public UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
47   ↳ 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 {
110     if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
111     {
112         _createSourceTreeMethods = () => new
113             ↳ LinksSourcesAvlBalancedTreeMethods<TLinkAddress>(Constants, _links, _header);
114     }
115 }

```



```

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

```

```

176         protected override ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress linkIndex) =>
            ↪ ref AsRef<RawLink<TLinkAddress>>(_links + (LinkSizeInBytes *
            ↪ ConvertToInt64(linkIndex)));
177     }
178 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Disposables;
5  using Platform.Singletons;
6  using Platform.Converters;
7  using Platform.Numbers;
8  using Platform.Memory;
9  using Platform.Data.Exceptions;
10 using Platform.Delegates;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Memory.United.Generic
15 {
16     /// <summary>
17     /// <para>
18     /// Represents the united memory links base.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <seealso cref="DisposableBase"/>
23     /// <seealso cref="ILinks{TLinkAddress}"/>
24     public abstract class UnitedMemoryLinksBase<TLinkAddress> : DisposableBase,
            ↪ ILinks<TLinkAddress>
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     {
143         _memory = memory;
144         _memoryReservationStep = memoryReservationStep;
145         Constants = constants;
146     }

```

```

144     /// <summary>
145     /// <para>
146     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
147     /// </para>
148     /// <para></para>
149     /// </summary>
150     /// <param name="memory">
151     /// <para>A memory.</para>
152     /// <para></para>
153     /// </param>
154     /// <param name="memoryReservationStep">
155     /// <para>A memory reservation step.</para>
156     /// <para></para>
157     /// </param>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
160     ↪ memoryReservationStep) : this(memory, memoryReservationStep,
161     ↪ Default<LinksConstants<TLinkAddress>>.Instance) { }
162
163     /// <summary>
164     /// <para>
165     /// Inits the memory.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="memory">
170     /// <para>The memory.</para>
171     /// <para></para>
172     /// </param>
173     /// <param name="memoryReservationStep">
174     /// <para>The memory reservation step.</para>
175     /// <para></para>
176     /// </param>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
179     {
180         if (memory.ReservedCapacity < memoryReservationStep)
181         {
182             memory.ReservedCapacity = memoryReservationStep;
183         }
184         SetPointers(memory);
185         ref var header = ref GetHeaderReference();
186         // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
187         memory.UsedCapacity = (Convert.ToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
188         ↪ LinkHeaderSizeInBytes;
189         // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
190         header.ReservedLinks = Convert.ToInt64((memory.ReservedCapacity -
191         ↪ LinkHeaderSizeInBytes) / LinkSizeInBytes);
192     }
193
194     /// <summary>
195     /// <para>
196     /// Counts the substitution.
197     /// </para>
198     /// <para></para>
199     /// </summary>
200     /// <param name="restriction">
201     /// <para>The substitution.</para>
202     /// <para></para>
203     /// </param>
204     /// <exception cref="NotSupportedException">
205     /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
206     /// <para></para>
207     /// </exception>
208     /// <returns>
209     /// <para>The link</para>
210     /// <para></para>
211     /// </returns>
212     [MethodImpl(MethodImplOptions.AggressiveInlining)]
213     public virtual TLinkAddress Count(IList<TLinkAddress>? restriction)
214     {
215         // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
216         if (restriction.Count == 0)
217         {
218             return Total;
219         }
220         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         {
292             return GetOne();
293         }
294         ref var storedLinkValue = ref GetLinkReference(index);
295         if (!AreEqual(source, any) && !AreEqual(target, any))

```

```

294         {
295             if (AreEqual(storedLinkValue.Source, source) &&
                ↪ AreEqual(storedLinkValue.Target, target))
296             {
297                 return GetOne();
298             }
299             return GetZero();
300         }
301         var value = default(TLinkAddress);
302         if (AreEqual(source, any))
303         {
304             value = target;
305         }
306         if (AreEqual(target, any))
307         {
308             value = source;
309         }
310         if (AreEqual(storedLinkValue.Source, value) ||
                ↪ AreEqual(storedLinkValue.Target, value))
311         {
312             return GetOne();
313         }
314         return GetZero();
315     }
316 }
317 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↪ поддерживаются.");
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,
    ↪ ReadHandler<TLinkAddress>? handler)
344 {
345     var constants = Constants;
346     var @break = constants.Break;
347     if (restriction.Count == 0)
348     {
349         for (var link = GetOne(); LessOrEqualThan(link,
                ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
350         {
351             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
352             {
353                 return @break;
354             }
355         }
356         return @break;
357     }
358     var @continue = constants.Continue;
359     var any = constants.Any;
360     var index = this.GetIndex(restriction);
361     if (restriction.Count == 1)
362     {
363         if (AreEqual(index, any))
364         {
365             return Each(Array.Empty<TLinkAddress>(), handler);
366         }

```

```

367         if (!Exists(index))
368         {
369             return @continue;
370         }
371         return handler(GetLinkStruct(index));
372     }
373     if (restriction.Count == 2)
374     {
375         var value = restriction[1];
376         if (AreEqual(index, any))
377         {
378             if (AreEqual(value, any))
379             {
380                 return Each(Array.Empty<TLinkAddress>(), handler);
381             }
382             if (AreEqual(Each(new Link<TLinkAddress>(index, value, any), handler),
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             {
441                 return handler(GetLinkStruct(index));
442             }
443             ref var storedLinkValue = ref GetLinkReference(index);
444             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 }
468 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↳ поддерживаются.");
469 }
470
471 /// <remarks>
472 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
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)]
516 public virtual TLinkAddress Create(IList<TLinkAddress>? substitution,
517     ↳ WriteHandler<TLinkAddress>? handler)
518 {
519     ref var header = ref GetHeaderReference();

```



```

514     var freeLink = header.FirstFreeLink;
515     if (!AreEqual(freeLink, Constants.Null))
516     {
517         UnusedLinksListMethods.Detach(freeLink);
518     }
519     else
520     {
521         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
522         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
523         {
524             throw new
525                 ↳ LinksLimitReachedException<TLinkAddress>(maximumPossibleInnerReference);
526         }
527         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
528         {
529             _memory.ReservedCapacity += _memory.ReservationStep;
530             SetPointers(_memory);
531             header = ref GetHeaderReference();
532             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
533                 ↳ LinkSizeInBytes);
534         }
535         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
536         _memory.UsedCapacity += LinkSizeInBytes;
537     }
538     return handler != null ? handler(null, new Link<TLinkAddress>(freeLink,
539         ↳ Constants.Null, Constants.Null)) : Constants.Continue;
540 }
541
542 /// <summary>
543 /// <para>
544 /// Deletes the substitution.
545 /// </para>
546 /// <para></para>
547 /// </summary>
548 /// <param name="restriction">
549 /// <para>The substitution.</para>
550 /// <para></para>
551 /// </param>
552 [MethodImpl(MethodImplOptions.AggressiveInlining)]
553 public virtual TLinkAddress Delete(IList<TLinkAddress>? restriction,
554     ↳ WriteHandler<TLinkAddress>? handler)
555 {
556     ref var header = ref GetHeaderReference();
557     var link = restriction[Constants.IndexPart];
558     var before = GetLinkStruct(link);
559     if (LessThan(link, header.AllocatedLinks))
560     {
561         UnusedLinksListMethods.AttachAsFirst(link);
562         // return handler?.Invoke(before, null);
563         return handler != null ? handler(before, null) : Constants.Continue;
564     }
565     else if (AreEqual(link, header.AllocatedLinks))
566     {
567         header.AllocatedLinks = Decrement(header.AllocatedLinks);
568         _memory.UsedCapacity -= LinkSizeInBytes;
569         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
570         // ↳ пока не дойдём до первой существующей связи
571         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
572         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
573             ↳ IsUnusedLink(header.AllocatedLinks))
574         {
575             UnusedLinksListMethods.Detach(header.AllocatedLinks);
576             header.AllocatedLinks = Decrement(header.AllocatedLinks);
577             _memory.UsedCapacity -= LinkSizeInBytes;
578         }
579         return handler != null ? handler(before, null) : Constants.Continue;
580     }
581     return Constants.Continue;
582 }
583
584 /// <summary>
585 /// <para>
586 /// Gets the link struct using the specified link index.
587 /// </para>
588 /// <para></para>
589 /// </summary>
590 /// <param name="linkIndex">
591 /// <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 ///
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>
660 /// <param name="link">
661 /// <para>The link.</para>
662 /// <para></para>
663 /// </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>
737     /// </param>
738     /// <param name="second">
739     /// <para>The second.</para>
740     /// <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;
814
815     /// <summary>
816     /// <para>
817     /// Determines whether this instance greater or equal than.

```

```

814    /// </para>
815    /// <para></para>
816    /// </summary>
817    /// <param name="first">
818    /// <para>The first.</para>
819    /// <para></para>
820    /// </param>
821    /// <param name="second">
822    /// <para>The second.</para>
823    /// <para></para>
824    /// </param>
825    /// <returns>
826    /// <para>The bool</para>
827    /// <para></para>
828    /// </returns>
829    [MethodImpl(MethodImplOptions.AggressiveInlining)]
830    protected virtual bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
831        ↪ _comparer.Compare(first, second) >= 0;
832
833    /// <summary>
834    /// <para>
835    /// <para>Converts the to int 64 using the specified value.
836    /// </para>
837    /// <para></para>
838    /// </summary>
839    /// <param name="value">
840    /// <para>The value.</para>
841    /// <para></para>
842    /// </param>
843    /// <returns>
844    /// <para>The long</para>
845    /// <para></para>
846    /// </returns>
847    [MethodImpl(MethodImplOptions.AggressiveInlining)]
848    protected virtual long ConvertToInt64(TLinkAddress value) =>
849        ↪ _addressToInt64Converter.Convert(value);
850
851    /// <summary>
852    /// <para>
853    /// <para>Converts the to address using the specified value.
854    /// </para>
855    /// <para></para>
856    /// </summary>
857    /// <param name="value">
858    /// <para>The value.</para>
859    /// <para></para>
860    /// </param>
861    /// <returns>
862    /// <para>The link</para>
863    /// <para></para>
864    /// </returns>
865    [MethodImpl(MethodImplOptions.AggressiveInlining)]
866    protected virtual TLinkAddress ConvertToAddress(long value) =>
867        ↪ _int64ToAddressConverter.Convert(value);
868
869    /// <summary>
870    /// <para>
871    /// <para>Adds the first.
872    /// </para>
873    /// <para></para>
874    /// </summary>
875    /// <param name="first">
876    /// <para>The first.</para>
877    /// <para></para>
878    /// </param>
879    /// <param name="second">
880    /// <para>The second.</para>
881    /// <para></para>
882    /// </param>
883    /// <returns>
884    /// <para>The link</para>
885    /// <para></para>
886    /// </returns>
887    [MethodImpl(MethodImplOptions.AggressiveInlining)]
888    protected virtual TLinkAddress Add(TLinkAddress first, TLinkAddress second) =>
889        ↪ Arithmetic<TLinkAddress>.Add(first, second);
890
891    /// <summary>

```

```

888     /// <para>
889     /// Subtracts the first.
890     /// </para>
891     /// <para></para>
892     /// </summary>
893     /// <param name="first">
894     /// <para>The first.</para>
895     /// <para></para>
896     /// </param>
897     /// <param name="second">
898     /// <para>The second.</para>
899     /// <para></para>
900     /// </param>
901     /// <returns>
902     /// <para>The link</para>
903     /// <para></para>
904     /// </returns>
905     [MethodImpl(MethodImplOptions.AggressiveInlining)]
906     protected virtual TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
907         ↪ Arithmetic<TLinkAddress>.Subtract(first, second);
908
909     /// <summary>
910     /// <para>
911     /// Increments the link.
912     /// </para>
913     /// <para></para>
914     /// </summary>
915     /// <param name="link">
916     /// <para>The link.</para>
917     /// <para></para>
918     /// </param>
919     /// <returns>
920     /// <para>The link</para>
921     /// <para></para>
922     /// </returns>
923     [MethodImpl(MethodImplOptions.AggressiveInlining)]
924     protected virtual TLinkAddress Increment(TLinkAddress link) =>
925         ↪ Arithmetic<TLinkAddress>.Increment(link);
926
927     /// <summary>
928     /// <para>
929     /// Decrements the link.
930     /// </para>
931     /// <para></para>
932     /// </summary>
933     /// <param name="link">
934     /// <para>The link.</para>
935     /// <para></para>
936     /// </param>
937     /// <returns>
938     /// <para>The link</para>
939     /// <para></para>
940     /// </returns>
941     [MethodImpl(MethodImplOptions.AggressiveInlining)]
942     protected virtual TLinkAddress Decrement(TLinkAddress link) =>
943         ↪ Arithmetic<TLinkAddress>.Decrement(link);
944
945     #region Disposable
946
947     /// <summary>
948     /// <para>
949     /// Gets the allow multiple dispose calls value.
950     /// </para>
951     /// <para></para>
952     /// </summary>
953     protected override bool AllowMultipleDisposeCalls
954     {
955         [MethodImpl(MethodImplOptions.AggressiveInlining)]
956         get => true;
957     }
958
959     /// <summary>
960     /// <para>
961     /// Disposes the manual.
962     /// </para>
963     /// <para></para>
964     /// </summary>
965     /// <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 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Methods.Lists;
3  using Platform.Converters;
4  using static System.Runtime.CompilerServices.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.United.Generic
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unused links list methods.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="AbsoluteCircularDoublyLinkedListMethods{TLinkAddress}"/>
17     /// <seealso cref="ILinksListMethods{TLinkAddress}"/>
18     public unsafe class UnusedLinksListMethods<TLinkAddress> :
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)]

```

```

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>
205    /// <param name="next">
206    /// <para>The next.</para>
207    /// <para></para>
208    /// </param>

```

```

204     /// </param>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override void SetNext(TLinkAddress element, TLinkAddress next) =>
        ↪ GetLinkReference(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 }

```

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

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.United
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLink<TLinkAddress> : IEquatable<RawLink<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<RawLink<TLinkAddress>>.Size;
27
28         /// <summary>
29         /// <para>
30         /// The source.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         public TLinkAddress Source;
35         /// <summary>
36         /// <para>
37         /// The target.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         public TLinkAddress Target;
42         /// <summary>
43         /// <para>
44         /// The left as source.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         public TLinkAddress LeftAsSource;
49         /// <summary>
50         /// <para>
51         /// The right as source.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         public TLinkAddress RightAsSource;
56         /// <summary>

```

```

57     /// <para>
58     /// The size as source.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLinkAddress SizeAsSource;
63     /// <summary>
64     /// <para>
65     /// The left as target.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public TLinkAddress LeftAsTarget;
70     /// <summary>
71     /// <para>
72     /// The right as target.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public TLinkAddress RightAsTarget;
77     /// <summary>
78     /// <para>
79     /// The size as target.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLinkAddress SizeAsTarget;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equals.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="obj">
92     /// <para>The obj.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override bool Equals(object obj) => obj is RawLink<TLinkAddress> link ?
    ↳ Equals(link) : false;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance equals.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="other">
109    /// <para>The other.</para>
110    /// <para></para>
111    /// </param>
112    /// <returns>
113    /// <para>The bool</para>
114    /// <para></para>
115    /// </returns>
116    [MethodImpl(MethodImplOptions.AggressiveInlining)]
117    public bool Equals(RawLink<TLinkAddress> other)
118        => _equalityComparer.Equals(Source, other.Source)
119        && _equalityComparer.Equals(Target, other.Target)
120        && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
121        && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
122        && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
123        && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
124        && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
125        && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);
126
127    /// <summary>
128    /// <para>
129    /// Gets the hash code.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// </returns>

```

```

134     /// <para>The int</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public override int GetHashCode() => (Source, Target, LeftAsSource, RightAsSource,
        ↳ SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
139
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     public static bool operator ==(RawLink<TLinkAddress> left, RawLink<TLinkAddress> right)
        ↳ => left.Equals(right);
142
143     [MethodImpl(MethodImplOptions.AggressiveInlining)]
144     public static bool operator !=(RawLink<TLinkAddress> left, RawLink<TLinkAddress> right)
        ↳ => !(left == right);
145 }
146 }

```

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

```

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

```

```

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>

```



```

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 }

```

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

```

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

```

```

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
58     /// <summary>
59     /// <para>
60     /// Gets the zero.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <returns>
65     /// <para>The uint</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override uint GetZero() => 0U;
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(uint value) => value == 0U;
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>

```

```

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,
        ↪ secondLink.Source, secondLink.Target);
360 }
361
362 /// <summary>
363 /// <para>
364 /// Determines whether this instance first is to the right of second.
365 /// </para>
366 /// <para></para>
367 /// </summary>
368 /// <param name="first">
369 /// <para>The first.</para>
370 /// <para></para>
371 /// </param>
372 /// <param name="second">
373 /// <para>The second.</para>
374 /// <para></para>
375 /// </param>
376 /// <returns>
377 /// <para>The bool</para>
378 /// <para></para>
379 /// </returns>
380 [MethodImpl(MethodImplOptions.AggressiveInlining)]
381 protected override bool FirstIsToTheRightOfSecond(uint first, uint second)
382 {
383     ref var firstLink = ref Links[first];
384     ref var secondLink = ref Links[second];
385     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
386 }
387
388 /// <summary>
389 /// <para>
390 /// Gets the header reference.
391 /// </para>
392 /// <para></para>
393 /// </summary>
394 /// <returns>
395 /// <para>A ref links header of uint</para>
396 /// <para></para>
397 /// </returns>
398 [MethodImpl(MethodImplOptions.AggressiveInlining)]
399 protected override ref LinksHeader<uint> GetHeaderReference() => ref *Header;
400
401 /// <summary>
402 /// <para>
403 /// Gets the link reference using the specified link.
404 /// </para>
405 /// <para></para>

```

```

406     /// </summary>
407     /// <param name="link">
408     /// <para>The link.</para>
409     /// <para></para>
410     /// </param>
411     /// <returns>
412     /// <para>A ref raw link of uint</para>
413     /// <para></para>
414     /// </returns>
415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     protected override ref RawLink<uint> GetLinkReference(uint link) => ref Links[link];
417 }
418 }

```

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

```

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

```

```

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

```

```

1.96 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs
1     using System.Runtime.CompilerServices;
2
3     #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

```

```

4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links sources size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksSourcesSizeBalancedTreeMethods :
15        ↳ UInt32LinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt32LinksSourcesSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="header">
32        /// <para>A header.</para>
33        /// <para></para>
34        /// </param>
35        public UInt32LinksSourcesSizeBalancedTreeMethods(LinksConstants<uint> constants,
36            ↳ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
37
38        /// <summary>
39        /// <para>
40        /// Gets the left reference using the specified node.
41        /// </para>
42        /// <para></para>
43        /// </summary>
44        /// <param name="node">
45        /// <para>The node.</para>
46        /// <para></para>
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>
78        /// </summary>
79        /// <param name="node">
80        /// <para>The node.</para>
81        /// <para></para>
82        /// </param>

```

```

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);

```

```

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

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

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links targets recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see
19        ↪ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">

```

```

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

```

```

98     /// <para>The uint</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override uint GetRight(uint node) => Links[node].RightAsTarget;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(uint node, uint left) => Links[node].LeftAsTarget = left;
120
121    /// <summary>
122    /// <para>
123    /// Sets the right using the specified node.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="node">
128    /// <para>The node.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="right">
132    /// <para>The right.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]
136    protected override void SetRight(uint node, uint right) => Links[node].RightAsTarget =
    ↪ 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">

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksTargetsSizeBalancedTreeMethods :
15         ↪ UInt32LinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt32LinksTargetsSizeBalancedTreeMethods(LinksConstants<uint> constants,
36             ↪ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
37
38         /// <summary>
39         /// <para>
40         /// Gets the left reference using the specified node.
41         /// </para>
42         /// <para></para>
43         /// </summary>

```

```

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

```

```

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

```

```

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

```

```

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 = OU;
277         link.RightAsTarget = OU;
278         link.SizeAsTarget = OU;
279     }
280 }
281 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Memory;
4  using Platform.Singletons;
5  using Platform.Data.Doublets.Memory.United.Generic;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     ///   ↳ organizing the storage of links with addresses represented as <see cref="uint" />.</para>
14     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
15     ///   ↳ размером, для организации хранения связей с адресами представленными в виде <see
16     ///   ↳ cref="uint"/>.</para>
17     /// </summary>
18     public unsafe class UInt32UnitedMemoryLinks : UnitedMemoryLinksBase<uint>
19     {
20         private readonly Func<ILinksTreeMethods<uint>> _createSourceTreeMethods;
21         private readonly Func<ILinksTreeMethods<uint>> _createTargetTreeMethods;
22         private LinksHeader<uint>* _header;
23         private RawLink<uint>* _links;
24
25         /// <summary>
26         /// <para>
27         ///   Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
28         /// </para>
29         /// <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         ///   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>
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         public UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
62         ///   ↳ DefaultLinksSizeStep) { }
63
64         /// <summary>

```

```

57     /// <para>
58     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="memory">
63     /// <para>A memory.</para>
64     /// <para></para>
65     /// </param>
66     /// <param name="memoryReservationStep">
67     /// <para>A memory reservation step.</para>
68     /// <para></para>
69     /// </param>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep) : this(memory, memoryReservationStep,
        ↳ Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }

72
73     /// <summary>
74     /// <para>
75     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <param name="memory">
80     /// <para>A memory.</para>
81     /// <para></para>
82     /// </param>
83     /// <param name="memoryReservationStep">
84     /// <para>A memory reservation step.</para>
85     /// <para></para>
86     /// </param>
87     /// <param name="constants">
88     /// <para>A constants.</para>
89     /// <para></para>
90     /// </param>
91     /// <param name="indexTreeType">
92     /// <para>A index tree type.</para>
93     /// <para></para>
94     /// </param>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep, LinksConstants<uint> constants, IndexTreeType indexTreeType)
        ↳ : base(memory, memoryReservationStep, constants)
97     {
98         if (indexTreeType == IndexTreeType.SizeBalancedTree)
99         {
100             _createSourceTreeMethods = () => new
                ↳ UInt32LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
101             _createTargetTreeMethods = () => new
                ↳ UInt32LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
102         }
103         else
104         {
105             _createSourceTreeMethods = () => new
                ↳ UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
                ↳ _header);
106             _createTargetTreeMethods = () => new
                ↳ UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
                ↳ _header);
107         }
108         Init(memory, memoryReservationStep);
109     }
110
111     /// <summary>
112     /// <para>
113     /// Sets the pointers using the specified memory.
114     /// </para>
115     /// <para></para>
116     /// </summary>
117     /// <param name="memory">
118     /// <para>The memory.</para>
119     /// <para></para>
120     /// </param>
121     [MethodImpl(MethodImplOptions.AggressiveInlining)]
122     protected override void SetPointers(IResizableDirectMemory memory)
123     {

```

```

124     _header = (LinksHeader<uint>*)memory.Pointer;
125     _links = (RawLink<uint>*)memory.Pointer;
126     SourcesTreeMethods = _createSourceTreeMethods();
127     TargetsTreeMethods = _createTargetTreeMethods();
128     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_links, _header);
129 }
130
131 /// <summary>
132 /// <para>
133 /// Resets the pointers.
134 /// </para>
135 /// <para></para>
136 /// </summary>
137 [MethodImpl(MethodImplOptions.AggressiveInlining)]
138 protected override void ResetPointers()
139 {
140     base.ResetPointers();
141     _links = null;
142     _header = null;
143 }
144
145 /// <summary>
146 /// <para>
147 /// Gets the header reference.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <returns>
152 /// <para>A ref links header of uint</para>
153 /// <para></para>
154 /// </returns>
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
157
158 /// <summary>
159 /// <para>
160 /// Gets the link reference using the specified link index.
161 /// </para>
162 /// <para></para>
163 /// </summary>
164 /// <param name="linkIndex">
165 /// <para>The link index.</para>
166 /// <para></para>
167 /// </param>
168 /// <returns>
169 /// <para>A ref raw link of uint</para>
170 /// <para></para>
171 /// </returns>
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 protected override ref RawLink<uint> GetLinkReference(uint linkIndex) => ref
    ↪ _links[linkIndex];
174
175 /// <summary>
176 /// <para>
177 /// Determines whether this instance are equal.
178 /// </para>
179 /// <para></para>
180 /// </summary>
181 /// <param name="first">
182 /// <para>The first.</para>
183 /// <para></para>
184 /// </param>
185 /// <param name="second">
186 /// <para>The second.</para>
187 /// <para></para>
188 /// </param>
189 /// <returns>
190 /// <para>The bool</para>
191 /// <para></para>
192 /// </returns>
193 [MethodImpl(MethodImplOptions.AggressiveInlining)]
194 protected override bool AreEqual(uint first, uint second) => first == second;
195
196 /// <summary>
197 /// <para>
198 /// Determines whether this instance less than.
199 /// </para>
200 /// <para></para>

```



```

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

```

```

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

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 32 unused links list methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UnusedLinksListMethods{uint}"/>
15    public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<uint>
16    {

```

```

17     private readonly RawLink<uint>* _links;
18     private readonly LinksHeader<uint>* _header;
19
20     /// <summary>
21     /// <para>
22     /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public UInt32UnusedLinksListMethods(RawLink<uint>* links, LinksHeader<uint>* header)
36         : base((byte*)links, (byte*)header)
37     {
38         _links = links;
39         _header = header;
40     }
41
42     /// <summary>
43     /// <para>
44     /// Gets the link reference using the specified link.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="link">
49     /// <para>The link.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>A ref raw link of uint</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref RawLink<uint> GetLinkReference(uint link) => ref _links[link];
58
59     /// <summary>
60     /// <para>
61     /// Gets the header reference.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <returns>
66     /// <para>A ref links header of uint</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
71 }
72 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using static System.Runtime.CompilerServices.Unsafe;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.United.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 links avl balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksAvlBalancedTreeMethodsBase{ulong}"/>
16     public unsafe abstract class UInt64LinksAvlBalancedTreeMethodsBase :
17         ↳ LinksAvlBalancedTreeMethodsBase<ulong>
18     {
19         /// <summary>
20         /// <para>
21         /// The links.

```

```

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,
52     ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
53     : base(constants, (byte*)links, (byte*)header)
54     {
55         Links = links;
56         Header = header;
57     }
58     /// <summary>
59     /// <para>
60     /// Gets the zero.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <returns>
65     /// <para>The ulong</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ulong GetZero() => 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 FirstIsToTheLeftOfSecond(ulong first, ulong second)
357    {
358        ref var firstLink = ref Links[first];
359        ref var secondLink = ref Links[second];
360        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
361            ↪ secondLink.Source, secondLink.Target);
362    }
363
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>
401    /// <returns>
402    /// <para>The ulong</para>
403    /// <para></para>
404    /// </returns>
405    [MethodImpl(MethodImplOptions.AggressiveInlining)]
406    protected override ulong GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;

```

```

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

```

```

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>
553    /// <returns>
554    /// <para>A ref raw link of ulong</para>
555    /// <para></para>
556    /// </returns>
557    [MethodImpl(MethodImplOptions.AggressiveInlining)]
558    protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];

```

```
554 }
555 }
```

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

```
1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 links recursionless size balanced tree methods base.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{ulong}" />
15    public unsafe abstract class UInt64LinksRecursionlessSizeBalancedTreeMethodsBase :
16    ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<ulong>
17    {
18        /// <summary>
19        /// <para>
20        /// The links.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        protected new readonly RawLink<ulong>* Links;
25        /// <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>
70        [MethodImpl(MethodImplOptions.AggressiveInlining)]
71        protected override ulong GetZero() => 0UL;
72
73        /// <summary>
74        /// <para>
75        /// Determines whether this instance equal to zero.
```

```

73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="value">
77     /// <para>The value.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The bool</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override bool EqualToZero(ulong value) => value == 0UL;
86
87     /// <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 FirstIsToLeftOfSecond(ulong first, ulong second)
356    {
357        ref var firstLink = ref Links[first];
358        ref var secondLink = ref Links[second];
359        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
360            ↪ secondLink.Source, secondLink.Target);
361    }
362
363    /// <summary>
364    /// <para>
365    /// Determines whether this instance first is to the right of second.
366    /// </para>
367    /// <para></para>
368    /// </summary>
369    /// <param name="first">
370    /// <para>The first.</para>
371    /// <para></para>
372    /// </param>
373    /// <param name="second">
374    /// <para>The second.</para>
375    /// <para></para>
376    /// </param>
377    /// <returns>
378    /// <para>The bool</para>
379    /// <para></para>
380    /// </returns>
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,
            ↪ secondLink.Source, secondLink.Target);
386     }
387
388     /// <summary>
389     /// <para>
390     /// Gets the header reference.
391     /// </para>
392     /// <para></para>
393     /// </summary>
394     /// <returns>
395     /// <para>A ref links header of ulong</para>
396     /// <para></para>
397     /// </returns>
398     [MethodImpl(MethodImplOptions.AggressiveInlining)]
399     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
400
401     /// <summary>
402     /// <para>
403     /// Gets the link reference using the specified link.
404     /// </para>
405     /// <para></para>
406     /// </summary>
407     /// <param name="link">
408     /// <para>The link.</para>
409     /// <para></para>
410     /// </param>
411     /// <returns>
412     /// <para>A ref raw link of ulong</para>
413     /// <para></para>
414     /// </returns>
415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
417 }
418 }

```

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

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 64 links sources avl balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
14    public unsafe class UInt64LinksSourcesAvlBalancedTreeMethods :
15        ↪ UInt64LinksAvlBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt64LinksSourcesAvlBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// </param>
26        /// <param name="links">
27        /// <para>A links.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="header">
31        /// <para>A header.</para>
32        /// <para></para>
33        /// </param>
34        public UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
35            ↪ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
36        {
37            ↪ { }
38
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)]
72            protected override ref ulong GetRightReference(ulong node) => ref
73            ↪ Links[node].RightAsSource;
74
75            /// <summary>
76            /// <para>
77            /// Gets the left using the specified node.
78            /// </para>
79            /// </summary>
80            /// <param name="node">
81            /// <para>The node.</para>
82            /// <para></para>
83            /// </param>
84            /// <returns>
85            /// <para>The ref ulong</para>
86            /// <para></para>
87            /// </returns>
88            [MethodImpl(MethodImplOptions.AggressiveInlining)]
89            protected override ref ulong GetLeftUsingNode(ulong node) => ref
90            ↪ Links[node].LeftUsingNode;
91
92            /// <summary>
93            /// <para>
94            /// Gets the right using the specified node.
95            /// </para>
96            /// </summary>
97            /// <param name="node">
98            /// <para>The node.</para>
99            /// <para></para>
100           /// </param>
101           /// <returns>
102           /// <para>The ref ulong</para>
103           /// <para></para>
104           /// </returns>
105           [MethodImpl(MethodImplOptions.AggressiveInlining)]
106           protected override ref ulong GetRightUsingNode(ulong node) => ref
107           ↪ Links[node].RightUsingNode;
108
109           /// <summary>
110           /// <para>
111           /// Gets the left using the specified node.
112           /// </para>
113           /// </summary>
114           /// <param name="node">
115           /// <para>The node.</para>
116           /// <para></para>
117           /// </param>
118           /// <returns>
119           /// <para>The ref ulong</para>
120           /// <para></para>
121           /// </returns>
122           [MethodImpl(MethodImplOptions.AggressiveInlining)]
123           protected override ref ulong GetLeftUsingNodeAndHeader(ulong node) => ref
124           ↪ Links[node].LeftUsingNodeAndHeader;
125
126           /// <summary>
127           /// <para>
128           /// Gets the right using the specified node.
129           /// </para>
130           /// </summary>
131           /// <param name="node">
132           /// <para>The node.</para>
133           /// <para></para>
134           /// </param>
135           /// <returns>
136           /// <para>The ref ulong</para>
137           /// <para></para>
138           /// </returns>
139           [MethodImpl(MethodImplOptions.AggressiveInlining)]
140           protected override ref ulong GetRightUsingNodeAndHeader(ulong node) => ref
141           ↪ Links[node].RightUsingNodeAndHeader;
142
143           /// <summary>
144           /// <para>
145           /// Gets the left using the specified node.
146           /// </para>
147           /// </summary>
148           /// <param name="node">
149           /// <para>The node.</para>
150           /// <para></para>
151           /// </param>
152           /// <returns>
153           /// <para>The ref ulong</para>
154           /// <para></para>
155           /// </returns>
156           [MethodImpl(MethodImplOptions.AggressiveInlining)]
157           protected override ref ulong GetLeftUsingNodeAndHeaderAndHeader(ulong node) => ref
158           ↪ Links[node].LeftUsingNodeAndHeaderAndHeader;
159
160           /// <summary>
161           /// <para>
162           /// Gets the right using the specified node.
163           /// </para>
164           /// </summary>
165           /// <param name="node">
166           /// <para>The node.</para>
167           /// <para></para>
168           /// </param>
169           /// <returns>
170           /// <para>The ref ulong</para>
171           /// <para></para>
172           /// </returns>
173           [MethodImpl(MethodImplOptions.AggressiveInlining)]
174           protected override ref ulong GetRightUsingNodeAndHeaderAndHeader(ulong node) => ref
175           ↪ Links[node].RightUsingNodeAndHeaderAndHeader;
176
177           /// <summary>
178           /// <para>
179           /// Gets the left using the specified node.
180           /// </para>
181           /// </summary>
182           /// <param name="node">
183           /// <para>The node.</para>
184           /// <para></para>
185           /// </param>
186           /// <returns>
187           /// <para>The ref ulong</para>
188           /// <para></para>
189           /// </returns>
190           [MethodImpl(MethodImplOptions.AggressiveInlining)]
191           protected override ref ulong GetLeftUsingNodeAndHeaderAndHeaderAndHeader(ulong node) => ref
192           ↪ Links[node].LeftUsingNodeAndHeaderAndHeaderAndHeader;
193
194           /// <summary>
195           /// <para>
196           /// Gets the right using the specified node.
197           /// </para>
198           /// </summary>
199           /// <param name="node">
200           /// <para>The node.</para>
201           /// <para></para>
202           /// </param>
203           /// <returns>
204           /// <para>The ref ulong</para>
205           /// <para></para>
206           /// </returns>
207           [MethodImpl(MethodImplOptions.AggressiveInlining)]
208           protected override ref ulong GetRightUsingNodeAndHeaderAndHeaderAndHeader(ulong node) => ref
209           ↪ Links[node].RightUsingNodeAndHeaderAndHeaderAndHeader;
210
211           /// <summary>
212           /// <para>
213           /// Gets the left using the specified node.
214           /// </para>
215           /// </summary>
216           /// <param name="node">
217           /// <para>The node.</para>
218           /// <para></para>
219           /// </param>
220           /// <returns>
221           /// <para>The ref ulong</para>
222           /// <para></para>
223           /// </returns>
224           [MethodImpl(MethodImplOptions.AggressiveInlining)]
225           protected override ref ulong GetLeftUsingNodeAndHeaderAndHeaderAndHeaderAndHeader(ulong node) => ref
226           ↪ Links[node].LeftUsingNodeAndHeaderAndHeaderAndHeaderAndHeader;
227
228           /// <summary>
229           /// <para>
230           /// Gets the right using the specified node.
231           /// </para>
232           /// </summary>
233           /// <param name="node">
234           /// <para>The node.</para>
235           /// <para></para>
236           /// </param>
237           /// <returns>
238           /// <para>The ref ulong</para>
239           /// <para></para>
240           /// </returns>
241           [MethodImpl(MethodImplOptions.AggressiveInlining)]
242           protected override ref ulong GetRightUsingNodeAndHeaderAndHeaderAndHeaderAndHeader(ulong node) => ref
243           ↪ Links[node].RightUsingNodeAndHeaderAndHeaderAndHeaderAndHeader;
244
245           /// <summary>
246           /// <para>
247           /// Gets the left using the specified node.
248           /// </para>
249           /// </summary>
250           /// <param name="node">
251           /// <para>The node.</para>
252           /// <para></para>
253           /// </param>
254           /// <returns>
255           /// <para>The ref ulong</para>
256           /// <para></para>
257           /// </returns>
258           [MethodImpl(MethodImplOptions.AggressiveInlining)]
259           protected override ref ulong GetLeftUsingNodeAndHeaderAndHeaderAndHeaderAndHeaderAndHeader(ulong node) => ref
260           ↪ Links[node].LeftUsingNodeAndHeaderAndHeaderAndHeaderAndHeaderAndHeader;
261
262           /// <summary>
263           /// <para>
264           /// Gets the right using the specified node.
265           /// </para>
266           /// </summary>
267           /// <param name="node">
268           /// <para>The node.</para>
269           /// <para></para>
270           /// </param>
271           /// <returns>
272           /// <para>The ref ulong</para>
273           /// <para></para>
274           /// </returns>
275           [MethodImpl(MethodImplOptions.AggressiveInlining)]
276           protected override ref ulong GetRightUsingNodeAndHeaderAndHeaderAndHeaderAndHeaderAndHeader(ulong node) => ref
277           ↪ Links[node].RightUsingNodeAndHeaderAndHeaderAndHeaderAndHeaderAndHeader;
278
279           /// <summary>
280           /// <para>
281           /// Gets the left using the specified node.
282           /// </para>
283           /// </summary>
284           /// <param name="node">
285           /// <para>The node.</para>
286           /// <para></para>
287           /// </param>
288           /// <returns>
289           /// &lt
```

```

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     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// </param>
238     /// <param name="value">
239     /// <para>The value.</para>
240     /// </param>
241     [MethodImpl(MethodImplOptions.AggressiveInlining)]
242     protected override void SetRightIsChild(ulong node, bool value) =>
243         ↳ SetRightIsChildValue(ref Links[node].SizeAsSource, 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">
252     /// <para>The node.</para>
253     /// <para></para>
254     /// </param>
255     /// <returns>
256     /// <para>The sbyte</para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override sbyte GetBalance(ulong node) =>
260         ↳ GetBalanceValue(Links[node].SizeAsSource);
261
262     /// <summary>
263     /// <para>
264     /// Sets the balance using the specified node.
265     /// </para>
266     /// <para></para>
267     /// </summary>
268     /// <param name="node">
269     /// <para>The node.</para>
270     /// <para></para>
271     /// </param>
272     /// <param name="value">
273     /// <para>The value.</para>
274     /// <para></para>
275     /// </param>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
278         ↳ Links[node].SizeAsSource, value);
279
280     /// <summary>
281     /// <para>
282     /// Gets the tree root.
283     /// </para>
284     /// <para></para>
285     /// </summary>
286     /// <returns>
287     /// <para>The ulong</para>
288     /// </returns>
289     [MethodImpl(MethodImplOptions.AggressiveInlining)]
290     protected override ulong GetTreeRoot() => Header->RootAsSource;
291
292     /// <summary>
293     /// <para>
294     /// Gets the base part value using the specified link.
295     /// </para>
296     /// <para></para>
297

```

```

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

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

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

/// <summary>

```

```

371     /// <para>
372     /// Clears the node using the specified node.
373     /// </para>
374     /// <para></para>
375     /// </summary>
376     /// <param name="node">
377     /// <para>The node.</para>
378     /// <para></para>
379     /// </param>
380     [MethodImpl(MethodImplOptions.AggressiveInlining)]
381     protected override void ClearNode(ulong node)
382     {
383         ref var link = ref Links[node];
384         link.LeftAsSource = OUL;
385         link.RightAsSource = OUL;
386         link.SizeAsSource = OUL;
387     }
388 }
389 }

```

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

```

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

```

```

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 FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
230     ↪     ulong secondSource, ulong secondTarget)
231     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
232     ↪     secondTarget);
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(ulong firstSource, ulong firstTarget,
262     ↪     ulong secondSource, ulong secondTarget)
263     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
264     ↪     secondTarget);
265
266     /// <summary>
267     /// <para>
268     /// <para>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;

```

```

279     }
280 }
281 }

```

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

```

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

```



```

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

```

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

```

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

```

```

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

```

```

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

```

```

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

171
172    /// <summary>
173    /// <para>
174    /// Determines whether this instance get left is child.
175    /// </para>
176    /// <para></para>
177    /// </summary>
178    /// <param name="node">
179    /// <para>The node.</para>
180    /// <para></para>
181    /// </param>
182    /// <returns>
183    /// <para>The bool</para>
184    /// <para></para>
185    /// </returns>
186    [MethodImpl(MethodImplOptions.AggressiveInlining)]
187    protected override bool GetLeftIsChild(ulong node) =>
    ↳ 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     /// </summary>
246     /// <param name="node">
247     /// <para>The node.</para>
248     /// </para>
249     /// </param>
250     /// <returns>
251     /// <para>The sbyte</para>
252     /// </para>
253     /// </returns>
254     [MethodImpl(MethodImplOptions.AggressiveInlining)]
255     protected override sbyte GetBalance(ulong node) =>
256         ↳ GetBalanceValue(Links[node].SizeAsTarget);
257
258     /// <summary>
259     /// <para>
260     /// Sets the balance using the specified node.
261     /// </para>
262     /// </summary>
263     /// <param name="node">
264     /// <para>The node.</para>
265     /// </para>
266     /// </param>
267     /// <param name="value">
268     /// <para>The value.</para>
269     /// </para>
270     /// </param>
271     [MethodImpl(MethodImplOptions.AggressiveInlining)]
272     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
273         ↳ Links[node].SizeAsTarget, value);
274
275     /// <summary>
276     /// <para>
277     /// Gets the tree root.
278     /// </para>
279     /// </summary>
280     /// <returns>
281     /// <para>The ulong</para>
282     /// </para>
283     /// </returns>
284     [MethodImpl(MethodImplOptions.AggressiveInlining)]
285     protected override ulong GetTreeRoot() => Header->RootAsTarget;
286
287     /// <summary>
288     /// <para>
289     /// Gets the base part value using the specified link.
290     /// </para>
291     /// </summary>
292     /// <param name="link">
293     /// <para>The link.</para>
294     /// </para>
295     /// </param>
296     /// <returns>
297     /// <para>The ulong</para>
298     /// </para>
299     /// </returns>
300     [MethodImpl(MethodImplOptions.AggressiveInlining)]
301     protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
302
303     /// <summary>
304     /// <para>
305     /// Determines whether this instance first is to the left of second.
306     /// </para>
307     /// </summary>
308     /// <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     {
381         ref var link = ref Links[node];
382         link.LeftAsTarget = OUL;
383         link.RightAsTarget = OUL;
384         link.SizeAsTarget = OUL;
385     }
386 }
387
388 }
389
390 }
```


1.108 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethodsBase.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 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>
72        [MethodImpl(MethodImplOptions.AggressiveInlining)]
73        protected override ref ulong GetRightReference(ulong node) => ref
74        ↳ Links[node].RightAsTarget;
```

```

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

```

```

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

```

```

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

```

1.109 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsSizeBalancedTreeMethods :
15         ↪ UInt64LinksSizeBalancedTreeMethodsBase

```

```

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

```

```

89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].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>
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     /// </summary>
177     /// </summary>
178     /// <returns>
179     /// <para>The ulong</para>
180     /// </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     /// </summary>
190     /// </summary>
191     /// <param name="link">
192     /// <para>The link.</para>
193     /// </para>
194     /// </param>
195     /// <returns>
196     /// <para>The ulong</para>
197     /// </para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override ulong GetBasePartValue(ulong link) => Links[link].Target;

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

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

```

```

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

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Memory;
4  using Platform.Singletons;
5  using Platform.Data.Doublets.Memory.United.Generic;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     ↪ organizing the storage of links with addresses represented as <see cref="ulong"
14     ↪ >/>.</para>
15     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
16     ↪ размером, для организации хранения связей с адресами представленными в виде <see
17     ↪ cref="ulong"/>.</para>
18     /// </summary>
19     public unsafe class UInt64UnitedMemoryLinks : UnitedMemoryLinksBase<ulong>
20     {
21         private readonly Func<ILinksTreeMethods<ulong>> _createSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<ulong>> _createTargetTreeMethods;
23         private LinksHeader<ulong>* _header;
24         private RawLink<ulong>* _links;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="address">
33         /// <para>A address.</para>

```



```

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">
99     /// <para>A index tree type.</para>
100    /// <para></para>
101    /// </param>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
104    /// → memoryReservationStep, LinksConstants<ulong> constants, IndexTreeType indexTreeType)
105    /// → : base(memory, memoryReservationStep, constants)
106    {

```

```

98     if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
99     {
100         _createSourceTreeMethods = () => new
101             ↳ UInt64LinksSourcesAvlBalancedTreeMethods(Constants, _links, _header);
102         _createTargetTreeMethods = () => new
103             ↳ UInt64LinksTargetsAvlBalancedTreeMethods(Constants, _links, _header);
104     }
105     else if (indexTreeType == IndexTreeType.SizeBalancedTree)
106     {
107         _createSourceTreeMethods = () => new
108             ↳ UInt64LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
109         _createTargetTreeMethods = () => new
110             ↳ UInt64LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
111     }
112     else
113     {
114         _createSourceTreeMethods = () => new
115             ↳ UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
116             ↳ _header);
117         _createTargetTreeMethods = () => new
118             ↳ UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
119             ↳ _header);
120     }
121     Init(memory, memoryReservationStep);
122 }
123
124 /// <summary>
125 /// <para>
126 /// Sets the pointers using the specified memory.
127 /// </para>
128 /// </summary>
129 /// <param name="memory">
130 /// <para>The memory.</para>
131 /// </param>
132 [MethodImpl(MethodImplOptions.AggressiveInlining)]
133 protected override void SetPointers(IResizableDirectMemory memory)
134 {
135     _header = (LinksHeader<ulong>*)memory.Pointer;
136     _links = (RawLink<ulong>*)memory.Pointer;
137     SourcesTreeMethods = _createSourceTreeMethods();
138     TargetsTreeMethods = _createTargetTreeMethods();
139     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
140 }
141
142 /// <summary>
143 /// <para>
144 /// Resets the pointers.
145 /// </para>
146 /// </summary>
147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
148 protected override void ResetPointers()
149 {
150     base.ResetPointers();
151     _links = null;
152     _header = null;
153 }
154
155 /// <summary>
156 /// <para>
157 /// Gets the header reference.
158 /// </para>
159 /// </summary>
160 /// <returns>
161 /// <para>A ref links header of ulong</para>
162 /// </returns>
163 [MethodImpl(MethodImplOptions.AggressiveInlining)]
164 protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
165
166 /// <summary>
167 /// <para>
168 /// Gets the link reference using the specified link index.
169 /// </para>
170 /// </summary>
171 /// <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
        ↪ _links[linkIndex];
179
180     /// <summary>
181     /// <para>
182     /// Determines whether this instance are equal.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <param name="first">
187     /// <para>The first.</para>
188     /// <para></para>
189     /// </param>
190     /// <param name="second">
191     /// <para>The second.</para>
192     /// <para></para>
193     /// </param>
194     /// <returns>
195     /// <para>The bool</para>
196     /// <para></para>
197     /// </returns>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     protected override bool AreEqual(ulong first, ulong second) => first == second;
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance less than.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="first">
208     /// <para>The first.</para>
209     /// <para></para>
210     /// </param>
211     /// <param name="second">
212     /// <para>The second.</para>
213     /// <para></para>
214     /// </param>
215     /// <returns>
216     /// <para>The bool</para>
217     /// <para></para>
218     /// </returns>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override bool LessThan(ulong first, ulong second) => first < second;
221
222     /// <summary>
223     /// <para>
224     /// Determines whether this instance less or equal than.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <param name="first">
229     /// <para>The first.</para>
230     /// <para></para>
231     /// </param>
232     /// <param name="second">
233     /// <para>The second.</para>
234     /// <para></para>
235     /// </param>
236     /// <returns>
237     /// <para>The bool</para>
238     /// <para></para>
239     /// </returns>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
242
243     /// <summary>
244     /// <para>

```

```

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

```

```

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

```

```

401     [MethodImpl(MethodImplOptions.AggressiveInlining)]
402     protected override ulong Increment(ulong link) => ++link;
403
404     /// <summary>
405     /// <para>
406     /// Decrements the link.
407     /// </para>
408     /// <para></para>
409     /// </summary>
410     /// <param name="link">
411     /// <para>The link.</para>
412     /// <para></para>
413     /// </param>
414     /// <returns>
415     /// <para>The ulong</para>
416     /// <para></para>
417     /// </returns>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     protected override ulong Decrement(ulong link) => --link;
420 }
421 }

```

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

```

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

```

```

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

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.PropertyOperators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the properties operator.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{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>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         public TLinkAddress GetValue(TLinkAddress @object, TLinkAddress property)
56         {
57             var links = _links;
58             var objectProperty = links.SearchOrDefault(@object, property);
59             if (_equalityComparer.Equals(objectProperty, default))
60             {
61                 return default;
62             }
63         }
64     }
65 }

```

```

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 }

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.PropertyOperators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the property operator.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLinkAddress}"/>
16     /// <seealso cref="IProperty{TLinkAddress, TLinkAddress}"/>
17     public class PropertyOperator<TLinkAddress> : LinksOperatorBase<TLinkAddress>,
18         ↳ IProperty<TLinkAddress, TLinkAddress>
19     {
20         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
21             ↳ EqualityComparer<TLinkAddress>.Default;
22         private readonly TLinkAddress _propertyMarker;
23         private readonly TLinkAddress _propertyValueMarker;
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="PropertyOperator"/> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="links">
32         /// <para>A links.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="propertyMarker">
36         /// <para>A property marker.</para>
37         /// <para></para>
38         /// </param>

```



```

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 continueConstant = 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 continueConstant;
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 }

```

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

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Collections.Stacks;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Stacks
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the stack.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLinkAddress}"/>
17     /// <seealso cref="IStack{TLinkAddress}"/>
18     public class Stack<TLinkAddress> : LinksOperatorBase<TLinkAddress>, IStack<TLinkAddress>
19     {
20         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         /// </param>
45         /// <param name="stack">
46         /// <para>A stack.</para>
47         /// </param>
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public Stack(ILinks<TLinkAddress> links, TLinkAddress stack) : base(links) => _stack =
50             ⇨ 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(),
105         ↪ _links.GetOrCreate(GetTop(), element));
106 }

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Stacks
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the stack extensions.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    public static class StackExtensions
14    {
15        /// <summary>
16        /// <para>
17        /// Creates the stack using the specified links.
18        /// </para>
19        /// <para></para>
20        /// </summary>
21        /// <typeparam name="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 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Data.Doublets;
5  using Platform.Delegates;
6  using Platform.Threading.Synchronization;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets
11 {
12     /// <remarks>
13     /// TODO: Autogeneration of synchronized wrapper (decorator).
14     /// TODO: Try to unfold code of each method using IL generation for performance improvements.
15     /// TODO: Or even to unfold multiple layers of implementations.
16     /// </remarks>
17     public class SynchronizedLinks<TLinkAddress> : ISynchronizedLinks<TLinkAddress>
18     {
19         /// <summary>
20         /// <para>
21         /// Gets the constants value.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public LinksConstants<TLinkAddress> Constants
26         {
27             [MethodImpl(MethodImplOptions.AggressiveInlining)]
28             get;
29         }
30
31         /// <summary>
32         /// <para>
33         /// Gets the sync root value.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         public ISynchronization SyncRoot
38         {
39             [MethodImpl(MethodImplOptions.AggressiveInlining)]
40             get;
41         }
42
43         /// <summary>
44         /// <para>
45         /// Gets the sync value.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         public ILinks<TLinkAddress> Sync
50         {
51             [MethodImpl(MethodImplOptions.AggressiveInlining)]
52             get;
53         }
54
55         /// <summary>
56         /// <para>
57         /// Gets the unsync value.
58         /// </para>

```

```

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

```

```

135     /// <para>The link address</para>
136     /// <para></para>
137     /// </returns>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     public TLinkAddress Each(ICollection<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
    ↪ handler) => SyncRoot.ExecuteReadOperation(restriction, handler, Unsync.Each);

140
141     /// <summary>
142     /// <para>
143     /// Creates the substitution.
144     /// </para>
145     /// <para></para>
146     /// </summary>
147     /// <param name="substitution">
148     /// <para>The substitution.</para>
149     /// <para></para>
150     /// </param>
151     /// <returns>
152     /// <para>The link address</para>
153     /// <para></para>
154     /// </returns>
155     [MethodImpl(MethodImplOptions.AggressiveInlining)]
156     public TLinkAddress Create(ICollection<TLinkAddress>? substitution,
    ↪ WriteHandler<TLinkAddress>? handler) => SyncRoot.ExecuteWriteOperation(substitution,
    ↪ handler, Unsync.Create);

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

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

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

```

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

```

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

```

```

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

```



```

149     /// <para></para>
150     /// </summary>
151     /// <param name="links">
152     /// <para>The links.</para>
153     /// <para></para>
154     /// </param>
155     /// <param name="sb">
156     /// <para>The sb.</para>
157     /// <para></para>
158     /// </param>
159     /// <param name="visited">
160     /// <para>The visited.</para>
161     /// <para></para>
162     /// </param>
163     /// <param name="linkIndex">
164     /// <para>The link index.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="isElement">
168     /// <para>The is element.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="appendElement">
172     /// <para>The append element.</para>
173     /// <para></para>
174     /// </param>
175     /// <param name="renderIndex">
176     /// <para>The render index.</para>
177     /// <para></para>
178     /// </param>
179     /// <param name="renderDebug">
180     /// <para>The render debug.</para>
181     /// <para></para>
182     /// </param>
183     /// <exception cref="ArgumentNullException">
184     /// <para></para>
185     /// <para></para>
186     /// </exception>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     public static void AppendStructure(this ILinks<ulong> links, StringBuilder sb,
        ↳ HashSet<ulong> visited, ulong linkIndex, Func<Link<ulong>, bool> isElement,
        ↳ Action<StringBuilder, Link<ulong>> appendElement, bool renderIndex = false, bool
        ↳ renderDebug = false)
189     {
190         if (sb == null)
191         {
192             throw new ArgumentNullException(nameof(sb));
193         }
194         if (linkIndex == Constants.Null || linkIndex == Constants.Any || linkIndex ==
        ↳ Constants.Itself)
195         {
196             return;
197         }
198         if (links.Exists(linkIndex))
199         {
200             if (visited.Add(linkIndex))
201             {
202                 sb.Append('(');
203                 var link = new Link<ulong>(links.GetLink(linkIndex));
204                 if (renderIndex)
205                 {
206                     sb.Append(link.Index);
207                     sb.Append(':');
208                 }
209                 if (link.Source == link.Index)
210                 {
211                     sb.Append(link.Index);
212                 }
213                 else
214                 {
215                     var source = new Link<ulong>(links.GetLink(link.Source));
216                     if (isElement(source))
217                     {
218                         appendElement(sb, source);
219                     }
220                     else
221                     {

```

```

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

```

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

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.IO;
5  using System.Runtime.CompilerServices;
6  using System.Threading;
7  using System.Threading.Tasks;
8  using Platform.Disposables;
9  using Platform.Timestamps;
10 using Platform.Unsafe;
11 using Platform.IO;
12 using Platform.Data.Doublets.Decorators;
13 using Platform.Delegates;
14 using Platform.Exceptions;
15 using TLinkAddress = System.UInt64;
16
17 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
18
19 namespace Platform.Data.Doublets
20 {
21     /// <summary>
22     /// <para>
23     /// Represents the int 64 links transactions layer.
24     /// </para>
25     /// <para></para>
26     /// </summary>
27     /// <seealso cref="LinksDisposableDecoratorBase{TLinkAddress}"/>
28     public class UInt64LinksTransactionsLayer : LinksDisposableDecoratorBase<TLinkAddress>
29     ↪    //-V3073
30     {
31         /// <remarks>
32         /// Альтернативные варианты хранения трансформации (элемента транзакции):
33         ///

```

```

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

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

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

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

246
247 /// <summary>
248 /// <para>
249 /// Determines whether this instance equals.
250 /// </para>
251 /// <para></para>
252 /// </summary>
253 /// <param name="other">
254 /// <para>The other.</para>

```

```

255     /// <para></para>
256     /// </param>
257     /// <returns>
258     /// <para>The bool</para>
259     /// <para></para>
260     /// </returns>
261     [MethodImpl(MethodImplOptions.AggressiveInlining)]
262     public bool Equals(Transition other) => TransactionId == other.TransactionId &&
        ↳ Before == other.Before && After == other.After && Timestamp == other.Timestamp;
263
264     [MethodImpl(MethodImplOptions.AggressiveInlining)]
265     public static bool operator ==(Transition left, Transition right) =>
        ↳ left.Equals(right);
266
267     [MethodImpl(MethodImplOptions.AggressiveInlining)]
268     public static bool operator !=(Transition left, Transition right) => !(left ==
        ↳ right);
269 }
270
271 /// <remarks>
272 /// Другие варианты реализации транзакций (атомарности):
273 /// 1. Разделение хранения значения связи ((Source Target) или (Source Linker
274   ↳ Target)) и индексов.
275 /// 2. Хранение трансформаций/операций в отдельном хранилище Links, но дополнительно
276   ↳ потребуется решить вопрос
277   ↳ со ссылками на внешние идентификаторы, или как-то иначе решить вопрос с
278   ↳ пересечениями идентификаторов.
279 ///
280 /// Где хранить промежуточный список транзакций?
281 ///
282 /// В оперативной памяти:
283 /// Минусы:
284 /// 1. Может усложнить систему, если она будет функционировать самостоятельно,
285   ↳ так как нужно отдельно выделять память под список трансформаций.
286 /// 2. Выделенной оперативной памяти может не хватить, в том случае,
287   ↳ если транзакция использует слишком много трансформаций.
288   ↳ -> Можно использовать жёсткий диск для слишком длинных транзакций.
289   ↳ -> Максимальный размер списка трансформаций можно ограничить / задать
290   ↳ константой.
291 /// 3. При подтверждении транзакции (Commit) все трансформации записываются разом
292   ↳ создавая задержку.
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 /// </remarks>
305 public class Transaction : DisposableBase
306 {
307     private readonly Queue<Transition> _transitions;
308     private readonly UInt64LinksTransactionsLayer _layer;
309     /// <summary>
310     /// <para>
311     /// Gets or sets the is committed value.
312     /// </para>
313     /// <para></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     /// <para></para>
321     /// </summary>
322     public bool IsReverted { get; private set; }
323     /// <summary>
324     /// <para>
325     /// Initializes a new <see cref="Transaction"/> instance.
326     /// </para>
327     /// <para></para>
328     /// </summary>
329     public Transaction()
330     {
331     }
332     public Transaction(UInt64LinksTransactionsLayer layer)
333     {
334         _layer = layer;
335     }
336     public Transaction(Queue<Transition> transitions)
337     {
338         _transitions = transitions;
339     }
340     public Transaction(Queue<Transition> transitions, UInt64LinksTransactionsLayer layer)
341     {
342         _transitions = transitions;
343         _layer = layer;
344     }
345     public void Commit()
346     {
347         if (IsCommitted)
348             return;
349         if (IsReverted)
350             Revert();
351         _layer.Commit(_transitions);
352         IsCommitted = true;
353     }
354     public void Revert()
355     {
356         if (!IsReverted)
357             return;
358         if (IsCommitted)
359             Commit();
360         _layer.Revert(_transitions);
361         IsReverted = true;
362     }
363     public void Dispose()
364     {
365         _transitions.Dispose();
366         _layer.Dispose();
367     }
368 }

```

```

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     /// <summary>

```

```

401     /// <para>
402     /// Ensures the transaction allows write operations using the specified transaction.
403     /// </para>
404     /// <para></para>
405     /// </summary>
406     /// <param name="transaction">
407     /// <para>The transaction.</para>
408     /// <para></para>
409     /// </param>
410     /// <exception cref="InvalidOperationException">
411     /// <para>Transation is committed.</para>
412     /// <para></para>
413     /// </exception>
414     /// <exception cref="InvalidOperationException">
415     /// <para>Transation is reverted.</para>
416     /// <para></para>
417     /// </exception>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
420     {
421         if (transaction.IsReverted)
422         {
423             throw new InvalidOperationException("Transation is reverted.");
424         }
425         if (transaction.IsCommitted)
426         {
427             throw new InvalidOperationException("Transation is committed.");
428         }
429     }
430
431     /// <summary>
432     /// <para>
433     /// Disposes the manual.
434     /// </para>
435     /// <para></para>
436     /// </summary>
437     /// <param name="manual">
438     /// <para>The manual.</para>
439     /// <para></para>
440     /// </param>
441     /// <param name="wasDisposed">
442     /// <para>The was disposed.</para>
443     /// <para></para>
444     /// </param>
445     [MethodImpl(MethodImplOptions.AggressiveInlining)]
446     protected override void Dispose(bool manual, bool wasDisposed)
447     {
448         if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
449         {
450             if (!IsCommitted && !IsReverted)
451             {
452                 Revert();
453             }
454             _layer.ResetCurrentTransation();
455         }
456     }
457 }
458
459 /// <summary>
460 /// <para>
461 /// The from seconds.
462 /// </para>
463 /// <para></para>
464 /// </summary>
465 public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
466 private readonly string _logAddress;
467 private readonly FileStream _log;
468 private readonly Queue<Transition> _transitions;
469 private readonly UniqueTimestampFactory _uniqueTimestampFactory;
470 private Task _transitionsPusher;
471 private Transition _lastCommittedTransition;
472 private TLinkAddress _currentTransactionId;
473 private Queue<Transition> _currentTransactionTransitions;
474 private Transaction _currentTransaction;
475 private TLinkAddress _lastCommittedTransactionId;
476
477 /// <summary>
478 /// <para>
479 /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.

```



```

480 /// </para>
481 /// <para></para>
482 /// </summary>
483 /// <param name="links">
484 /// <para>A links.</para>
485 /// <para></para>
486 /// </param>
487 /// <param name="logAddress">
488 /// <para>A log address.</para>
489 /// <para></para>
490 /// </param>
491 /// <exception cref="ArgumentNullException">
492 /// <para></para>
493 /// <para></para>
494 /// </exception>
495 /// <exception cref="NotSupportedException">
496 /// <para>Database is damaged, autorecovery is not supported yet.</para>
497 /// <para></para>
498 /// </exception>
499 [MethodImpl(MethodImplOptions.AggressiveInlining)]
500 public UInt64LinksTransactionsLayer(ILinks<TLinkAddress> links, string logAddress)
501     : base(links)
502 {
503     if (string.IsNullOrEmpty(logAddress))
504     {
505         throw new ArgumentNullException(nameof(logAddress));
506     }
507     // В первой строке файла хранится последняя законченная транзакция.
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>
548 /// <para></para>
549 /// </returns>
550 [MethodImpl(MethodImplOptions.AggressiveInlining)]
551 public IList<TLinkAddress> GetLinkValue(TLinkAddress link) => _links.GetLink(link);
552
553 /// <summary>
554 /// <para>
555 /// Creates the substitution.
556 /// </para>
557 /// <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    /// <summary>
576    /// <para>
577    /// Updates the substitution.
578    /// </para>
579    /// <para></para>
580    /// </summary>
581    /// <param name="restriction">
582    /// <para>The substitution.</para>
583    /// <para></para>
584    /// </param>
585    /// <param name="substitution">
586    /// <para>The substitution.</para>
587    /// <para></para>
588    /// </param>
589    /// <returns>
590    /// <para>The link index.</para>
591    /// <para></para>
592    /// </returns>
593    [MethodImpl(MethodImplOptions.AggressiveInlining)]
594    public override TLinkAddress Update(IList<TLinkAddress>? restriction,
    ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
595    {
596        return _links.Update(restriction, substitution, (before, after) =>
597        {
598            CommitTransition(new Transition(_uniqueTimestampFactory,
599            ↪ _currentTransactionId, new Link<TLinkAddress>(before), new
600            ↪ Link<TLinkAddress>(after)));
601            return handler != null ? handler(before, after) : Constants.Continue;
602        });
603    }
604    /// <summary>
605    /// <para>
606    /// Deletes the substitution.
607    /// </para>
608    /// <para></para>
609    /// </summary>
610    /// <param name="restriction">
611    /// <para>The substitution.</para>
612    /// <para></para>
613    /// </param>
614    [MethodImpl(MethodImplOptions.AggressiveInlining)]
615    public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
    ↪ WriteHandler<TLinkAddress>? handler)
616    {
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    }
625    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

625 private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
    ↳ _transitions;
626 [MethodImpl(MethodImplOptions.AggressiveInlining)]
627 private void CommitTransition(Transition transition)
628 {
629     if (_currentTransaction != null)
630     {
631         Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
632     }
633     var transitions = GetCurrentTransitions();
634     transitions.Enqueue(transition);
635 }
636 [MethodImpl(MethodImplOptions.AggressiveInlining)]
637 private void RevertTransition(Transition transition)
638 {
639     if (transition.After.IsNull()) // Revert Deletion with Creation
640     {
641         _links.Create();
642     }
643     else if (transition.Before.IsNull()) // Revert Creation with Deletion
644     {
645         _links.Delete(transition.After.Index);
646     }
647     else // Revert Update
648     {
649         _links.Update(new[] { transition.After.Index, transition.Before.Source,
    ↳ transition.Before.Target });
650     }
651 }
652 [MethodImpl(MethodImplOptions.AggressiveInlining)]
653 private void ResetCurrentTransation()
654 {
655     _currentTransactionId = 0;
656     _currentTransactionTransitions = null;
657     _currentTransaction = null;
658 }
659 [MethodImpl(MethodImplOptions.AggressiveInlining)]
660 private void PushTransitions()
661 {
662     if (_log == null || _transitions == null)
663     {
664         return;
665     }
666     for (var i = 0; i < _transitions.Count; i++)
667     {
668         var transition = _transitions.Dequeue();
669
670         _log.Write(transition);
671         _lastCommittedTransition = transition;
672     }
673 }
674 [MethodImpl(MethodImplOptions.AggressiveInlining)]
675 private void TransitionsPusher()
676 {
677     while (!Disposable.IsDisposed && _transitionsPusher != null)
678     {
679         Thread.Sleep(DefaultPushDelay);
680         PushTransitions();
681     }
682 }
683
684 /// <summary>
685 /// <para>
686 /// Begins the transaction.
687 /// </para>
688 /// <para></para>
689 /// </summary>
690 /// <returns>
691 /// <para>The transaction</para>
692 /// <para></para>
693 /// </returns>
694 [MethodImpl(MethodImplOptions.AggressiveInlining)]
695 public Transaction BeginTransaction() => new Transaction(this);
696 [MethodImpl(MethodImplOptions.AggressiveInlining)]
697 private void DisposeTransitions()
698 {
699     try
700     {

```

```

701         var pusher = _transitionsPusher;
702         if (pusher != null)
703         {
704             _transitionsPusher = null;
705             pusher.Wait();
706         }
707         if (_transitions != null)
708         {
709             PushTransitions();
710         }
711         _log.DisposeIfPossible();
712         FileHelpers.WriteFirst(_logAddress, _lastCommittedTransition);
713     }
714     catch (Exception ex)
715     {
716         ex.Ignore();
717     }
718 }
719
720 #region DisposalBase
721
722 /// <summary>
723 /// <para>
724 /// Disposes the manual.
725 /// </para>
726 /// <para></para>
727 /// </summary>
728 /// <param name="manual">
729 /// <para>The manual.</para>
730 /// <para></para>
731 /// </param>
732 /// <param name="wasDisposed">
733 /// <para>The was disposed.</para>
734 /// <para></para>
735 /// </param>
736 [MethodImpl(MethodImplOptions.AggressiveInlining)]
737 protected override void Dispose(bool manual, bool wasDisposed)
738 {
739     if (!wasDisposed)
740     {
741         DisposeTransitions();
742     }
743     base.Dispose(manual, wasDisposed);
744 }
745
746 #endregion
747 }
748 }

```

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

```

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

```

```

30     }
31
32     [Fact]
33     public static void MultipleRandomCreationsAndDeletionsTest()
34     {
35         Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
36             ↳ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
37             ↳ implementation of tree cuts out 5 bits from the address space.
38         Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
39             ↳ stMultipleRandomCreationsAndDeletions(100));
40         Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
41             ↳ MultipleRandomCreationsAndDeletions(100));
42         Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
43             ↳ tMultipleRandomCreationsAndDeletions(100));
44     }
45     private static void Using<TLinkAddress>(Action<ILinks<TLinkAddress>> action)
46     {
47         var unitedMemoryLinks = new UnitedMemoryLinks<TLinkAddress>(new
48             ↳ HeapResizableDirectMemory());
49         using (var logFile = File.Open("linksLogger.txt", FileMode.Create, FileAccess.Write))
50         {
51             LoggingDecorator<TLinkAddress> links = new(unitedMemoryLinks, logFile);
52             action(links);
53         }
54
55         File.Delete("db.links");
56         using var ffiLinks = new FFI.UnitedMemoryLinks<TLinkAddress>("db.links");
57         action(ffiLinks);
58     }
59 }
60

```

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

```

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

```

```

47         links.DeleteAllUsages(root);
48     }
49     Assert.Equal(3U, links.Count());
50 }
51 }
52 }
53 }

```

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

```

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

```

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

```

1  using System.IO;
2  using Xunit;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.United.Specific;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public static class ResizableDirectMemoryLinksTests
10     {
11         private static readonly LinksConstants<ulong> _constants =
12             ↪ Default<LinksConstants<ulong>>.Instance;
13
14         [Fact]
15         public static void BasicFileMappedMemoryTest()
16         {
17             var tempFilename = Path.GetTempFileName();
18             using (var memoryAdapter = new UInt64UnitedMemoryLinks(tempFilename))
19             {
20                 memoryAdapter.TestBasicMemoryOperations();
21             }
22             File.Delete(tempFilename);
23         }
24
25         [Fact]
26         public static void BasicHeapMemoryTest()
27         {
28             using (var memory = new
29                 ↪ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
30             using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
31                 ↪ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
32             {
33                 memoryAdapter.TestBasicMemoryOperations();
34             }
35         }
36
37         private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
38         {
39             var link = memoryAdapter.Create();
40             memoryAdapter.Delete(link);
41         }
42
43         [Fact]
44         public static void NonexistentReferencesHeapMemoryTest()
45         {
46             using (var memory = new
47                 ↪ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))

```

```

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

```

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

```

1  using Xunit;
2  using Platform.Scopes;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Reflection;
6  using Platform.Data.Doublets.Memory.United.Generic;
7  using Platform.Data.Doublets.Memory.United.Specific;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class ScopeTests
12     {
13         [Fact]
14         public static void SingleDependencyTest()
15         {
16             using (var scope = new Scope())
17             {
18                 scope.IncludeAssemblyOf<IMemory>();
19                 var instance = scope.Use<IDirectMemory>();
20                 Assert.IsType<HeapResizableDirectMemory>(instance);
21             }
22         }
23
24         [Fact]
25         public static void CascadeDependencyTest()
26         {
27             using (var scope = new Scope())
28             {
29                 scope.Include<TemporaryFileMappedResizableDirectMemory>();
30                 scope.Include<UInt64UnitedMemoryLinks>();
31                 var instance = scope.Use<ILinks<ulong>>();
32                 Assert.IsType<UInt64UnitedMemoryLinks>(instance);
33             }
34         }
35
36         [Fact(Skip = "Would be fixed later.")]
37         public static void FullAutoResolutionTest()
38         {
39             using (var scope = new Scope(autoInclude: true, autoExplore: true))
40             {
41                 var instance = scope.Use<UInt64Links>();
42                 Assert.IsType<UInt64Links>(instance);
43             }
44         }
45
46         [Fact]
47         public static void TypeParametersTest()
48         {
49             using (var scope = new Scope<Types<HeapResizableDirectMemory,
50             ↪ UnitedMemoryLinks<ulong>>>())
51             {
52                 var links = scope.Use<ILinks<ulong>>();
53                 Assert.IsType<UnitedMemoryLinks<ulong>>(links);
54             }
55         }
56     }
57 }

```

```

55     }
56 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Generic;
5  using Platform.Data.Doublets.Memory;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryGenericLinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using<byte>(links => links.TestCRUDOperations());
15             Using<ushort>(links => links.TestCRUDOperations());
16             Using<uint>(links => links.TestCRUDOperations());
17             Using<ulong>(links => links.TestCRUDOperations());
18         }
19
20         [Fact]
21         public static void RawNumbersCRUDTest()
22         {
23             UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
24             UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
25             UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
26             UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
27         }
28
29         [Fact]
30         public static void MultipleRandomCreationsAndDeletionsTest()
31         {
32             Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
33                 ↪ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
34                 ↪ implementation of tree cuts out 5 bits from the address space.
35             Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
36                 ↪ stMultipleRandomCreationsAndDeletions(100));
37             Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
38                 ↪ MultipleRandomCreationsAndDeletions(100));
39             Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
40                 ↪ tMultipleRandomCreationsAndDeletions(100));
41         }
42         private static void Using<TLinkAddress>(Action<ILinks<TLinkAddress>> action)
43         {
44             using (var dataMemory = new HeapResizableDirectMemory())
45             using (var indexMemory = new HeapResizableDirectMemory())
46             using (var memory = new SplitMemoryLinks<TLinkAddress>(dataMemory, indexMemory))
47             {
48                 action(memory);
49             }
50         }
51         private static void
52             ↪ UsingWithExternalReferences<TLinkAddress>(Action<ILinks<TLinkAddress>> action)
53         {
54             var contants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
55                 ↪ true);
56             using (var dataMemory = new HeapResizableDirectMemory())
57             using (var indexMemory = new HeapResizableDirectMemory())
58             using (var memory = new SplitMemoryLinks<TLinkAddress>(dataMemory, indexMemory,
59                 ↪ SplitMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, contants))
60             {
61                 action(memory);
62             }
63         }
64     }
65 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLinkAddress = System.UInt32;
6
7  namespace Platform.Data.Doublets.Tests

```



```

8 {
9     public unsafe static class SplitMemoryUInt32LinksTests
10    {
11        [Fact]
12        public static void CRUDTest()
13        {
14            Using(links => links.TestCRUDOperations());
15        }
16
17        [Fact]
18        public static void RawNumbersCRUDTest()
19        {
20            UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
21        }
22
23        [Fact]
24        public static void MultipleRandomCreationsAndDeletionsTest()
25        {
26            Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(500));
27        }
28        private static void Using(Action<ILinks<TLinkAddress>> action)
29        {
30            using (var dataMemory = new HeapResizableDirectMemory())
31            using (var indexMemory = new HeapResizableDirectMemory())
32            using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory))
33            {
34                action(memory);
35            }
36        }
37        private static void UsingWithExternalReferences(Action<ILinks<TLinkAddress>> action)
38        {
39            var constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
40                true);
41            using (var dataMemory = new HeapResizableDirectMemory())
42            using (var indexMemory = new HeapResizableDirectMemory())
43            using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory,
44                UInt32SplitMemoryLinks.DefaultLinksSizeStep, constants))
45            {
46                action(memory);
47            }
48        }
49    }
50 }

```

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

```

1 using System;
2 using Xunit;
3 using Platform.Memory;
4 using Platform.Data.Doublets.Memory.Split.Specific;
5 using TLinkAddress = System.UInt64;
6
7 namespace Platform.Data.Doublets.Tests
8 {
9     public unsafe static class SplitMemoryUInt64LinksTests
10    {
11        [Fact]
12        public static void CRUDTest()
13        {
14            Using(links => links.TestCRUDOperations());
15        }
16
17        [Fact]
18        public static void RawNumbersCRUDTest()
19        {
20            UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
21        }
22
23        [Fact]
24        public static void MultipleRandomCreationsAndDeletionsTest()
25        {
26            Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(500));
27        }
28        private static void Using(Action<ILinks<TLinkAddress>> action)
29        {
30            using (var dataMemory = new HeapResizableDirectMemory())
31            using (var indexMemory = new HeapResizableDirectMemory())

```

```

32         using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory))
33         {
34             action(memory);
35         }
36     }
37     private static void UsingWithExternalReferences(Action<ILinks<TLinkAddress>> action)
38     {
39         var constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
40             ↪ true);
41         using (var dataMemory = new HeapResizableDirectMemory())
42         using (var indexMemory = new HeapResizableDirectMemory())
43         using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
44             ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, constants))
45         {
46             action(memory);
47         }
48     }

```

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

```

1  using System.Collections.Generic;
2  using Xunit;
3  using Platform.Ranges;
4  using Platform.Numbers;
5  using Platform.Random;
6  using Platform.Setters;
7  using Platform.Converters;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class TestExtensions
12     {
13         public static void TestCRUDOperations<T>(this ILinks<T> links)
14         {
15             var constants = links.Constants;
16
17             var equalityComparer = EqualityComparer<T>.Default;
18
19             var zero = default(T);
20             var one = Arithmetic.Increment(zero);
21
22             // Create Link
23             Assert.True(equalityComparer.Equals(links.Count(), zero));
24
25             var setter = new Setter<T>(constants.Null);
26             links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
27
28             Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
29
30             var linkAddress = links.Create();
31
32             var link = new Link<T>(links.GetLink(linkAddress));
33
34             Assert.True(link.Count == 3);
35             Assert.True(equalityComparer.Equals(link.Index, linkAddress));
36             Assert.True(equalityComparer.Equals(link.Source, constants.Null));
37             Assert.True(equalityComparer.Equals(link.Target, constants.Null));
38
39             Assert.True(equalityComparer.Equals(links.Count(), one));
40
41             // Get first link
42             setter = new Setter<T>(constants.Null);
43             links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
44
45             Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
46
47             // Update link to reference itself
48             links.Update(linkAddress, linkAddress, linkAddress);
49
50             link = new Link<T>(links.GetLink(linkAddress));
51
52             Assert.True(equalityComparer.Equals(link.Source, linkAddress));
53             Assert.True(equalityComparer.Equals(link.Target, linkAddress));
54
55             // Update link to reference null (prepare for delete)
56             var updated = links.Update(linkAddress, constants.Null, constants.Null);
57
58             Assert.True(equalityComparer.Equals(updated, linkAddress));
59

```

```

60     link = new Link<T>(links.GetLink(linkAddress));
61
62     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
63     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
64
65     // Delete link
66     links.Delete(linkAddress);
67
68     Assert.True(equalityComparer.Equals(links.Count(), zero));
69
70     setter = new Setter<T>(constants.Null);
71     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
72
73     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
74 }
75
76 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
77 {
78     // Constants
79     var constants = links.Constants;
80     var equalityComparer = EqualityComparer<T>.Default;
81
82     var zero = default(T);
83     var one = Arithmetic.Increment(zero);
84     var two = Arithmetic.Increment(one);
85
86     var h106E = new Hybrid<T>(106L, isExternal: true);
87     var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
88     var h108E = new Hybrid<T>(-108L);
89
90     Assert.Equal(106L, h106E.AbsoluteValue);
91     Assert.Equal(107L, h107E.AbsoluteValue);
92     Assert.Equal(108L, h108E.AbsoluteValue);
93
94     // Create Link (External -> External)
95     var linkAddress1 = links.Create();
96
97     links.Update(linkAddress1, h106E, h108E);
98
99     var link1 = new Link<T>(links.GetLink(linkAddress1));
100
101     Assert.True(equalityComparer.Equals(link1.Source, h106E));
102     Assert.True(equalityComparer.Equals(link1.Target, h108E));
103
104     // Create Link (Internal -> External)
105     var linkAddress2 = links.Create();
106
107     links.Update(linkAddress2, linkAddress1, h108E);
108
109     var link2 = new Link<T>(links.GetLink(linkAddress2));
110
111     Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
112     Assert.True(equalityComparer.Equals(link2.Target, h108E));
113
114     // Create Link (Internal -> Internal)
115     var linkAddress3 = links.Create();
116
117     links.Update(linkAddress3, linkAddress1, linkAddress2);
118
119     var link3 = new Link<T>(links.GetLink(linkAddress3));
120
121     Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
122     Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
123
124     // Search for created link
125     var setter1 = new Setter<T>(constants.Null);
126     links.Each(h106E, h108E, setter1.SetAndReturnFalse);
127
128     Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
129
130     // Search for nonexistent link
131     var setter2 = new Setter<T>(constants.Null);
132     links.Each(h106E, h107E, setter2.SetAndReturnFalse);
133
134     Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
135
136     // Update link to reference null (prepare for delete)
137     var updated = links.Update(linkAddress3, constants.Null, constants.Null);
138
139     Assert.True(equalityComparer.Equals(updated, linkAddress3));

```

```

140     link3 = new Link<T>(links.GetLink(linkAddress3));
141
142     Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
143     Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
144
145     // Delete link
146     links.Delete(linkAddress3);
147
148     Assert.True(equalityComparer.Equals(links.Count(), two));
149
150     var setter3 = new Setter<T>(constants.Null);
151     links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
152
153     Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
154 }
155
156 public static void TestMultipleCreationsAndDeletions<TLinkAddress>(this
157     ↪ ILinks<TLinkAddress> links, int numberOfOperations)
158 {
159     for (int i = 0; i < numberOfOperations; i++)
160     {
161         links.Create();
162     }
163     for (int i = 0; i < numberOfOperations; i++)
164     {
165         links.Delete(links.Count());
166     }
167 }
168
169 public static void TestMultipleRandomCreationsAndDeletions<TLinkAddress>(this
170     ↪ ILinks<TLinkAddress> links, int maximumOperationsPerCycle)
171 {
172     var comparer = Comparer<TLinkAddress>.Default;
173     var addressToUInt64Converter = CheckedConverter<TLinkAddress, ulong>.Default;
174     var uint64ToAddressConverter = CheckedConverter<ulong, TLinkAddress>.Default;
175     for (var N = 1; N < maximumOperationsPerCycle; N++)
176     {
177         var random = new System.Random(N);
178         var created = 0UL;
179         var deleted = 0UL;
180         for (var i = 0; i < N; i++)
181         {
182             var linksCount = addressToUInt64Converter.Convert(links.Count());
183             var createPoint = random.NextBoolean();
184             if (linksCount >= 2 && createPoint)
185             {
186                 var linksAddressRange = new Range<ulong>(1, linksCount);
187                 TLinkAddress source = uint64ToAddressConverter.Convert(random.NextUInt64 ↪
188                     ↪ (linksAddressRange));
189                 TLinkAddress target = uint64ToAddressConverter.Convert(random.NextUInt64 ↪
190                     ↪ (linksAddressRange));
191                 ↪ //-V3086
192                 var resultLink = links.GetOrCreate(source, target);
193                 if (comparer.Compare(resultLink,
194                     ↪ uint64ToAddressConverter.Convert(linksCount)) > 0)
195                 {
196                     created++;
197                 }
198             }
199             else
200             {
201                 links.Create();
202                 created++;
203             }
204         }
205         Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
206         for (var i = 0; i < N; i++)
207         {
208             TLinkAddress link = uint64ToAddressConverter.Convert((ulong)i + 1UL);
209             if (links.Exists(link))
210             {
211                 links.Delete(link);
212                 deleted++;
213             }
214         }
215         Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
216     }
217 }

```

```

212     }
213 }
214 }

```

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

```

1  using Platform.Data.Doublets.Memory;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Doublets.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
            ↳ Platform.IO.TemporaryFile());
15
16         public static ILinks<TLinkAddress> CreateLinks<TLinkAddress>(string dataDBFilename)
17         {
18             var linksConstants = new
19                 ↳ LinksConstants<TLinkAddress>(enableExternalReferencesSupport: true);
20             return new UnitedMemoryLinks<TLinkAddress>(new
21                 ↳ FileMappedResizableDirectMemory(dataDBFilename),
22                 ↳ UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
23                 ↳ IndexTreeType.Default);
24         }
25         [Fact]
26         public void FormatStructureWithExternalReferenceTest()
27         {
28             ILinks<TLinkAddress> links = CreateLinks();
29             TLinkAddress zero = default;
30             var one = Arithmetic.Increment(zero);
31             var markerIndex = one;
32             var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
33             var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
34                 ↳ markerIndex));
35             AddressToRawNumberConverter<TLinkAddress> addressToNumberConverter = new();
36             var numberAddress = addressToNumberConverter.Convert(1);
37             var numberLink = links.GetOrCreate(numberMarker, numberAddress);
38             var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
39                 ↳ true);
40             Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
41         }
42     }
43 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLinkAddress = System.UInt32;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public unsafe static class UnitedMemoryUInt32LinksTests
12     {
13         [Fact]
14         public static void CRUDTest()
15         {
16             Using(links => links.TestCRUDOperations());
17         }
18
19         [Fact]
20         public static void RawNumbersCRUDTest()
21         {
22             Using(links => links.TestRawNumbersCRUDOperations());
23         }
24
25         [Fact]
26         public static void MultipleRandomCreationsAndDeletionsTest()
27         {
28             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
29                 ↳ leRandomCreationsAndDeletions(100));
30         }
31     }
32 }

```

```

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

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLinkAddress = System.UInt64;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public unsafe static class UnitedMemoryUInt64LinksTests
12     {
13         [Fact]
14         public static void CRUDTest()
15         {
16             Using(links => links.TestCRUDOperations());
17         }
18
19         [Fact]
20         public static void RawNumbersCRUDTest()
21         {
22             Using(links => links.TestRawNumbersCRUDOperations());
23         }
24
25         [Fact]
26         public static void MultipleRandomCreationsAndDeletionsTest()
27         {
28             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
29         }
30         private static void Using(Action<ILinks<TLinkAddress>> action)
31         {
32             using (var scope = new Scope<Types<HeapResizableDirectMemory,
33             ↳ UInt64UnitedMemoryLinks>>())
34             {
35                 action(scope.Use<ILinks<TLinkAddress>>());
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/NonNullContentsLinkDeletionResolver.cs`, 15
`./csharp/Platform.Data.Doublets/Decorators/UInt32Links.cs`, 16
`./csharp/Platform.Data.Doublets/Decorators/UInt64Links.cs`, 18
`./csharp/Platform.Data.Doublets/Decorators/UniLinks.cs`, 20
`./csharp/Platform.Data.Doublets/Doublet.cs`, 25
`./csharp/Platform.Data.Doublets/DoubletComparer.cs`, 28
`./csharp/Platform.Data.Doublets/FFI/UInt32UnitedMemoryLinks.cs`, 28
`./csharp/Platform.Data.Doublets/FFI/UnitedMemoryLinks.cs`, 30
`./csharp/Platform.Data.Doublets/ILinks.cs`, 39
`./csharp/Platform.Data.Doublets/ILinksExtensions.cs`, 40
`./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs`, 60
`./csharp/Platform.Data.Doublets/Link.cs`, 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`, 87
`./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
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 174
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 178

./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 182
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs, 186
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 190
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 195
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 201
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 202
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 206
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 210
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 213
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 217
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs, 224
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 225
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase.cs, 231
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 237
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods.cs, 241
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 245
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethods.cs, 248
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 252
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.cs, 258
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 264
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 265
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 268
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 272
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 276
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 279
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 286
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 287
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 296
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 303
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 309
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 314
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 318
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 322
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 327
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 331
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs, 335
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs, 338
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs, 351
./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs, 354
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 356
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 361
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 367
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs, 370
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 374
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 378
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 382
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 387
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 388
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 396
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 401
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 406
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 412
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 416
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 419
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 424
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 428
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 432
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs, 438
./csharp/Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs, 439
./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs, 440
./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, 446
./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 450