

## 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 = restriction[_constants.IndexPart];
47         // Use Facade (the last decorator) to ensure recursion working correctly
48         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
49         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
50         return handlerState.Result;
51     }
52 }

```

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

```

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

```

```

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(ICollection<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(ICollection<TLinkAddress>? restriction,
162     ↪ ICollection<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(ICollection<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 link = restriction[_constants.IndexPart];
102         var links = _links;
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(substitution[constants.SourcePart],
58             ↪ substitution[constants.TargetPart]);
59             return links.Update(restriction, substitution, handler);
60         }
61     }
62 }

```

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

```

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

```

```

22     /// </para>
23     /// <para></para>
24     /// </summary>
25     /// <param name="links">
26     /// <para>A links.</para>
27     /// <para></para>
28     /// </param>
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     public LinksNullConstantToSelfReferenceResolver(ILinks<TLinkAddress> links) :
        ↪ base(links) { }
31
32     /// <summary>
33     /// <para>
34     /// Creates the substitution.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="substitution">
39     /// <para>The substitution.</para>
40     /// <para></para>
41     /// </param>
42     /// <returns>
43     /// <para>The link</para>
44     /// <para></para>
45     /// </returns>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public override TLinkAddress Create(ICollection<TLinkAddress>? substitution,
        ↪ WriteHandler<TLinkAddress>? handler)
48     {
49         ↪ return _links.CreatePoint(handler);
50     }
51
52     /// <summary>
53     /// <para>
54     /// Updates the substitution.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     /// <param name="restriction">
59     /// <para>The substitution.</para>
60     /// <para></para>
61     /// </param>
62     /// <param name="substitution">
63     /// <para>The substitution.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public override TLinkAddress Update(ICollection<TLinkAddress>? restriction,
        ↪ ICollection<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler) =>
        ↪ _links.Update(restriction, _links.ResolveConstantAsSelfReference(_constants.Null,
        ↪ restriction, substitution), handler);
72 }
73 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links uniqueness resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLinkAddress}"/>
17     public class LinksUniquenessResolver<TLinkAddress> : LinksDecoratorBase<TLinkAddress>
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(substitution[constants.SourcePart],
59 ↪ substitution[constants.TargetPart]);
60     if (_equalityComparer.Equals(newLinkAddress, default))
61     {
62         return links.Update(restriction, substitution, handler);
63     }
64     return ResolveAddressChangeConflict(restriction[constants.IndexPart],
65 ↪ newLinkAddress, handler);
66 }
67
68 /// <summary>
69 /// <para>
70 /// Resolves the address change conflict using the specified old link address.
71 /// </para>
72 /// <para></para>
73 /// </summary>
74 /// <param name="oldLinkAddress">
75 /// <para>The old link address.</para>
76 /// <para></para>
77 /// </param>
78 /// <param name="newLinkAddress">
79 /// <para>The new link address.</para>
80 /// <para></para>
81 /// </param>
82 /// <returns>
83 /// <para>The new link address.</para>
84 /// <para></para>
85 /// </returns>
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 protected virtual 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

```

```

92     }
93 }

```

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

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links usages validator.

```

```

13  /// </para>
14  /// <para></para>
15  /// </summary>
16  /// <seealso cref="LinksDecoratorBase{TLinkAddress}"/>
17  public class LinksUsagesValidator<TLinkAddress> : LinksDecoratorBase<TLinkAddress>
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(restriction[_constants.IndexPart]);
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 link = restriction[_constants.IndexPart];
74          var links = _links;
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         if (handlerState.Handler != null)
26         {
27             handlerState.Apply(handlerState.Handler(before, after));
28         }
29         _logStreamWriter.WriteLine($"Create. Before: {new Link<TLinkAddress>(before)}.
30         ↪ After: {new Link<TLinkAddress>(after)}");
31         return _constants.Continue;
32     });
33 }
34
35 public override TLinkAddress Update(IList<TLinkAddress>? restriction,
36     ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
37 {
38     WriteHandlerState<TLinkAddress> handlerState = new(_constants.Continue,
39     ↪ _constants.Break, handler);
40     return base.Update(restriction, substitution, (before, after) =>
41     {
42         if (handlerState.Handler != null)
43         {
44             handlerState.Apply(handlerState.Handler(before, after));
45         }
46         _logStreamWriter.WriteLine($"Update. Before: {new Link<TLinkAddress>(before)}.
47         ↪ After: {new Link<TLinkAddress>(after)}");
48         return _constants.Continue;
49     });
50 }
51
52 public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
53     ↪ WriteHandler<TLinkAddress>? handler)
54 {
55     WriteHandlerState<TLinkAddress> handlerState = new(_constants.Continue,
56     ↪ _constants.Break, handler);
57     return base.Delete(restriction, (before, after) =>
58     {
59         if (handlerState.Handler != null)
60         {
61             handlerState.Apply(handlerState.Handler(before, after));
62         }
63         _logStreamWriter.WriteLine($"Delete. Before: {new Link<TLinkAddress>(before)}.
64         ↪ After: {new Link<TLinkAddress>(after)}");
65         return _constants.Continue;
66     });
67 }
68 }
69 }
70 }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Delegates;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Decorators
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the non null contents link deletion resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLinkAddress}"/>

```

```

17 public class NonNullContentsLinkDeletionResolver<TLinkAddress> :
    ↳ LinksDecoratorBase<TLinkAddress>
18 {
19     /// <summary>
20     /// <para>
21     /// Initializes a new <see cref="NonNullContentsLinkDeletionResolver"/> instance.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     /// <param name="links">
26     /// <para>A links.</para>
27     /// <para></para>
28     /// </param>
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     public NonNullContentsLinkDeletionResolver(ILinks<TLinkAddress> links) : base(links) { }
31
32     /// <summary>
33     /// <para>
34     /// Deletes the restriction.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="restriction">
39     /// <para>The restriction.</para>
40     /// <para></para>
41     /// </param>
42     [MethodImpl(MethodImplOptions.AggressiveInlining)]
43     public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
    ↳ WriteHandler<TLinkAddress>? handler)
44     {
45         var linkIndex = restriction[_constants.IndexPart];
46         var constants = _links.Constants;
47         WriteHandlerState<TLinkAddress> handlerResult = new(constants.Continue,
    ↳ constants.Break, handler);
48         handlerResult.Apply(_links.EnforceResetValues(linkIndex, handlerResult.Handler));
49         handlerResult.Apply(_links.Delete(restriction, handlerResult.Handler));
50         return handlerResult.Result;
51     }
52 }
53 }

```

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

```

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

```



```

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

```

```

112     /// </param>
113     [MethodImpl(MethodImplOptions.AggressiveInlining)]
114     public override TLinkAddress Delete(IList<TLinkAddress>? restriction,
115         ↳ WriteHandler<TLinkAddress>? handler)
116     {
117         var linkIndex = restriction[_constants.IndexPart];
118         var constants = _links.Constants;
119         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
120             ↳ constants.Break, handler);
121         handlerState.Apply(_links.EnforceResetValues(linkIndex, handlerState.Handler));
122         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
123         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
124         return handlerState.Result;
125     }
126 }

```

## 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         /// </summary>
45         /// <param name="links">
46         /// <para>A links.</para>
47         /// </param>
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public UInt64Links(ILinks<TLinkAddress> links) : base(links) { }
50
51         /// <summary>
52         /// <para>
53         /// Creates the substitution.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         /// <param name="substitution">
58         /// <para>The substitution.</para>
59         /// </param>
60     }
61 }

```

```

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

```

```

127     {
128         var linkIndex = restriction[_constants.IndexPart];
129         var constants = _links.Constants;
130         WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
131             ↪ constants.Break, handler);
132         handlerState.Apply(_links.EnforceResetValues(linkIndex, handlerState.Handler));
133         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
134         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
135         return handlerState.Result;
136     }
137 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Platform.Collections;
5  using Platform.Collections.Lists;
6  using Platform.Data.Universal;
7  using Platform.Delegates;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Decorators
12 {
13     /// <remarks>
14     /// What does empty pattern (for condition or substitution) mean? Nothing or Everything?
15     /// Now we go with nothing. And nothing is something one, but empty, and cannot be changed
16     ↪ by itself. But can cause creation (update from nothing) or deletion (update to nothing).
17     ///
18     /// TODO: Decide to change to IDoubletLinks or not to change. (Better to create
19     ↪ DefaultUniLinksBase, that contains logic itself and can be implemented using both
20     ↪ IDoubletLinks and ILinks.)
21     /// </remarks>
22     internal class UniLinks<TLinkAddress> : LinksDecoratorBase<TLinkAddress>,
23     ↪ IUniLinks<TLinkAddress>
24     {
25         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
26         ↪ EqualityComparer<TLinkAddress>.Default;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="UniLinks"/> instance.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="links">
35         /// <para>A links.</para>
36         /// <para></para>
37         /// </param>
38         public UniLinks(ILinks<TLinkAddress> links) : base(links) { }
39         private struct Transition
40         {
41             /// <summary>
42             /// <para>
43             /// The before.
44             /// </para>
45             /// <para></para>
46             /// </summary>
47             public IList<TLinkAddress>? Before;
48             /// <summary>
49             /// <para>
50             /// The after.
51             /// </para>
52             /// <para></para>
53             /// </summary>
54             public IList<TLinkAddress>? After;
55
56             /// <summary>
57             /// <para>
58             /// Initializes a new <see cref="Transition"/> instance.
59             /// </para>
60             /// <para></para>
61             /// </summary>
62             /// <param name="before">
63             /// <para>A before.</para>
64             /// <para></para>
65             /// </param>

```

```

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

```

```

131     /// var newValue = Memory.GetLinkValue(link);
132     /// newValue[Constants.IndexPart] = Constants.Itself;
133     /// newValue[Constants.SourcePart] =
    ↳ Equals(substitution[Constants.SourcePart], Constants.Itself) ?
    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.SourcePart];
134     /// newValue[Constants.TargetPart] =
    ↳ Equals(substitution[Constants.TargetPart], Constants.Itself) ?
    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.TargetPart];
135     /// var matchDecision = matchedHandler(matchedLink, newValue);
136     /// if (Equals(matchDecision, Constants.Break))
137     ///     return false;
138     /// if (!Equals(matchDecision, Constants.Skip))
139     ///     transitions.Add(new Transition(matchedLink, newValue));
140     /// return true;
141     /// };
142     /// if (!Memory.Each(handler, restriction))
143     ///     return Constants.Break;
144     /// }
145     /// }
146     /// 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 //    {
213 //        matchedLinks = new List<IList<T>>();
214 //        Func<T, bool> handler = link =>
215 //        {
216 //            var matchedLink = Memory.GetLinkValue(link);
217 //            var matchDecision = matchedHandler(matchedLink);
218 //            if (Equals(matchDecision, Constants.Break))
219 //                return false;
220 //            if (!Equals(matchDecision, Constants.Skip))
221 //                matchedLinks.Add(matchedLink);
222 //            return true;
223 //        };
224 //        if (!Memory.Each(handler, restriction))
225 //            return Constants.Break;
226 //    }
227 //    if (!matchedLinks.IsNullOrEmpty())
228 //    {
229 //        var totalMatchedLinks = matchedLinks.Count;
230 //        for (var i = 0; i < totalMatchedLinks; i++)
231 //        {
232 //            var matchedLink = matchedLinks[i];
233 //            if (substitutedHandler != null)
234 //            {
235 //                var newValue = new List<T>(); // TODO: Prepare value to update here
236 //                // TODO: Decide is it actually needed to use Before and After
237 //                ↳ substitution handling.
238 //                var substitutedDecision = substitutedHandler(matchedLink,
239 //                ↳ newValue);
240 //                if (Equals(substitutedDecision, Constants.Break))
241 //                    return Constants.Break;
242 //                if (Equals(substitutedDecision, Constants.Continue))
243 //                {
244 //                    // Actual update here
245 //                    Memory.SetLinkValue(newValue);
246 //                }
247 //                if (Equals(substitutedDecision, Constants.Skip))
248 //                {
249 //                    // Cancel the update. TODO: decide use separate Cancel
250 //                    ↳ constant or Skip is enough?
251 //                }
252 //            }
253 //        }
254 //    }
255 //}
256 return _constants.Continue;
257 }
258
259 /// <summary>
260 /// <para>
261 /// Triggers the pattern or condition.
262 /// </para>
263 /// <para></para>
264 /// </summary>
265 /// <param name="patternOrCondition">
266 /// <para>The pattern or condition.</para>
267 /// <para></para>
268 /// </param>
269 /// <param name="matchHandler">
270 /// <para>The match handler.</para>
271 /// <para></para>
272 /// </param>
273 /// <param name="substitution">
274 /// <para>The substitution.</para>
275 /// <para></para>
276 /// </param>
277 /// <param name="substitutionHandler">
278 /// <para>The substitution handler.</para>
279 /// <para></para>
280 /// </param>

```

```

277     /// <exception cref="NotImplementedException">
278     /// <para></para>
279     /// <para></para>
280     /// </exception>
281     /// <exception cref="NotSupportedException">
282     /// <para></para>
283     /// <para></para>
284     /// </exception>
285     /// <exception cref="NotSupportedException">
286     /// <para></para>
287     /// <para></para>
288     /// </exception>
289     /// <exception cref="NotSupportedException">
290     /// <para></para>
291     /// <para></para>
292     /// </exception>
293     /// <exception cref="NotSupportedException">
294     /// <para></para>
295     /// <para></para>
296     /// </exception>
297     /// <returns>
298     /// <para>The link</para>
299     /// <para></para>
300     /// </returns>
301 public TLinkAddress Trigger(ICollection<TLinkAddress>? patternOrCondition,
    ↪ ReadHandler<TLinkAddress>? matchHandler, ICollection<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.Equals(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 = (ICollection<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
357     {
358         throw new NotSupportedException();
359     }
360 }
361 else // Replace / Update
362 {
363     if (patternOrCondition.Count == 1) //-V3125
364     {
365         var linkToUpdate = patternOrCondition[0];
366         var before = _links.GetLink(linkToUpdate);
367         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
            ↪ constants.Break))
368         {
369             return constants.Break;
370         }
371         var after = (IList<TLinkAddress>?)substitution.ToArray(); //-V3125
372         if (_equalityComparer.Equals(after[0], default))
373         {
374             after[0] = linkToUpdate;
375         }
376         if (substitution.Count == 1)
377         {
378             if (!_equalityComparer.Equals(substitution[0], linkToUpdate))
379             {
380                 after = _links.GetLink(substitution[0]);
381                 _links.Update(linkToUpdate, constants.Null, constants.Null);
382                 _links.Delete(linkToUpdate);
383             }
384         }
385         else if (substitution.Count == 3)
386         {
387             //Links.Update(after);
388         }
389         else
390         {
391             throw new NotSupportedException();
392         }
393         return matchHandler != null ? substitutionHandler(before, after) :
            ↪ constants.Continue;
394     }
395     else
396     {
397         throw new NotSupportedException();
398     }
399 }
400 }
401
402 /// <remarks>
403 /// IList[IList[IList[T]]]
404 /// |         |         |         ||
405 /// |         |         - - - - - ||
406 /// |         |         link    ||
407 /// |         - - - - - - - - - |
408 /// |         change          |
409 /// - - - - - - - - - -
410 ///         changes
411 /// </remarks>
412 public IList<IList<IList<TLinkAddress>?>> Trigger(IList<TLinkAddress>? condition,
    ↪ IList<TLinkAddress>? substitution)
413 {
414     var changes = new List<IList<IList<TLinkAddress>?>>();
415     var @continue = _constants.Continue;
416     Trigger(condition, AlwaysContinue, substitution, (before, after) =>
417     {
418         var change = new[] { before, after };
419         changes.Add(change);
420         return @continue;
421     });

```

```

422         return changes;
423     }
424     private TLinkAddress AlwaysContinue(ICollection<TLinkAddress>? linkToMatch) =>
        ↪ _constants.Continue;
425 }
426 }

```

## 1.18 ./csharp/Platform.Data.Doublets/Doublet.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets
8  {
9
10     /// <summary>
11     /// <para>.</para>
12     /// <para>.</para>
13     /// </summary>
14     /// <typeparam>
15     /// <para>.</para>
16     /// <para>.</para>
17     /// </typeparam>
18     public struct Doublet<T> : IEquatable<Doublet<T>>
19     {
20         private static readonly EqualityComparer<T> _equalityComparer =
            ↪ EqualityComparer<T>.Default;
21
22         /// <summary>
23         /// <para>.</para>
24         /// <para>.</para>
25         /// </summary>
26         /// <typeparam name="T">
27         /// <para>.</para>
28         /// <para>.</para>
29         /// </typeparam>
30         public readonly T Source;
31
32         /// <summary>
33         /// <para>.</para>
34         /// <para>.</para>
35         /// </summary>
36         /// <typeparam name="T">
37         /// <para>.</para>
38         /// <para>.</para>
39         /// </typeparam>
40         public readonly T Target;
41
42         /// <summary>
43         /// <para>.</para>
44         /// <para>.</para>
45         /// </summary>
46         /// <typeparam name="T">
47         /// <para>.</para>
48         /// <para>.</para>
49         /// </typeparam>
50         /// <param name="source">
51         /// <para>.</para>
52         /// <para>.</para>
53         /// </param>
54         /// <param name="target">
55         /// <para>.</para>
56         /// <para>.</para>
57         /// </param>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public Doublet(T source, T target)
60         {
61             Source = source;
62             Target = target;
63         }
64
65         /// <summary>
66         /// <para>.</para>
67         /// <para>.</para>
68         /// </summary>
69         /// <returns>
70         /// <para>.</para>

```

```

71     /// <para>.</para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     public override string ToString() => $"{Source}->{Target}";
75
76     /// <summary>
77     /// <para>.</para>
78     /// <para>.</para>
79     /// </summary>
80     /// <typeparam>
81     /// <para>.</para>
82     /// <para>.</para>
83     /// </typeparam>
84     /// <param name="other">
85     /// <para>.</para>
86     /// <para>.</para>
87     /// </param>
88     /// <returns>
89     /// <para>.</para>
90     /// <para>.</para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public bool Equals(Doublet<T> other) => _equalityComparer.Equals(Source, other.Source)
94     ↪ && _equalityComparer.Equals(Target, other.Target);
95
96     /// <summary>
97     /// <para>.</para>
98     /// <para>.</para>
99     /// </summary>
100    /// <typeparam>
101    /// <para>.</para>
102    /// <para>.</para>
103    /// </typeparam>
104    /// <param name="obj">
105    /// <para>.</para>
106    /// <para>.</para>
107    /// </param>
108    /// <returns>
109    /// <para>.</para>
110    /// <para>.</para>
111    /// </returns>
112    [MethodImpl(MethodImplOptions.AggressiveInlining)]
113    public override bool Equals(object obj) => obj is Doublet<T> doublet ?
114    ↪ base.Equals(doublet) : false;
115
116    /// <summary>
117    /// <para>.</para>
118    /// <para>.</para>
119    /// </summary>
120    /// <returns>
121    /// <para>.</para>
122    /// <para>.</para>
123    /// </returns>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    public override int GetHashCode() => (Source, Target).GetHashCode();
126
127    /// <summary>
128    /// <para>.</para>
129    /// <para>.</para>
130    /// </summary>
131    /// <param name="left">
132    /// <para>.</para>
133    /// <para>.</para>
134    /// </param>
135    /// <param name="right">
136    /// <para>.</para>
137    /// <para>.</para>
138    /// </param>
139    /// <returns>
140    /// <para>.</para>
141    /// <para>.</para>
142    /// </returns>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    public static bool operator ==(Doublet<T> left, Doublet<T> right) => left.Equals(right);
145
146    /// <summary>
147    /// <para>.</para>
148    /// <para>.</para>
149    /// </summary>

```

```

147     /// </summary>
148     /// <param name="left">
149     /// <para>.</para>
150     /// <para>.</para>
151     /// </param>
152     /// <param name="right">
153     /// <para>.</para>
154     /// <para>.</para>
155     /// </param>
156     /// <returns>
157     /// <para>.</para>
158     /// <para>.</para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public static bool operator !=(Doublet<T> left, Doublet<T> right) => !(left == right);
162 }
163 }

```

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

```

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

```

```
60 }
```

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

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.InteropServices;
5 using Platform.Converters;
6 using Platform.Delegates;
7 using Platform.Disposables;
8
9 namespace Platform.Data.Doublets.FFI
10 {
11     using 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?.Invoke(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?.Invoke(new
67                             ↪ Link<TLinkAddress>(before.Index, before.Source, before.Target), new
68                             ↪ 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         }
79     }
80 }
```

```

66     }
67 }
68
69 public TLinkAddress Update(IList<TLinkAddress>? restriction, IList<TLinkAddress>?
    ↳ substitution, WriteHandler<TLinkAddress>? handler)
70 {
71     unsafe
72     {
73         var restrictionArray = stackalloc uint[restriction.Count];
74         for (var i = 0; i < restriction.Count; i++)
75         {
76             restrictionArray[i] = restriction[i];
77         }
78         var substitutionArray = stackalloc uint[substitution.Count];
79         for (var i = 0; i < substitution.Count; i++)
80         {
81             substitutionArray[i] = substitution[i];
82         }
83         Methods.UpdateCallback_UInt32 callback = (before, after) => handler?.Invoke(new
            ↳ Link<TLinkAddress>(before.Index, before.Source, before.Target), new
            ↳ Link<TLinkAddress>(after.Index, after.Source, after.Target)) ??
            ↳ Constants.Continue;
84         return Methods.UInt32UnitedMemoryLinks_Update(_ptr, restrictionArray,
            ↳ (nuint)(restriction?.Count ?? 0), substitutionArray,
            ↳ (nuint)(substitution?.Count ?? 0), callback);
85     }
86 }
87
88 public TLinkAddress Delete(IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
    ↳ handler)
89 {
90     unsafe
91     {
92         var restrictionArray = stackalloc uint[restriction.Count];
93         for (var i = 0; i < restriction.Count; i++)
94         {
95             restrictionArray[i] = restriction[i];
96         }
97         Methods.DeleteCallback_UInt32 callback = (before, after) => handler?.Invoke(new
            ↳ Link<TLinkAddress>(before.Index, before.Source, before.Target), new
            ↳ Link<TLinkAddress>(after.Index, after.Source, after.Target)) ??
            ↳ Constants.Continue;
98         return Methods.UInt32UnitedMemoryLinks_Delete(_ptr, restrictionArray,
            ↳ (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 }

```

```

89     public delegate UInt64 DeleteCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
90         ↪ after);
91
92     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
93     public static extern void* ByteUnitedMemoryLinks_New(string path);
94
95     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
96     public static extern void* UInt16UnitedMemoryLinks_New(string path);
97
98     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
99     public static extern void* UInt32UnitedMemoryLinks_New(string path);
100
101     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
102     public static extern void* UInt64UnitedMemoryLinks_New(string path);
103
104     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
105     public static extern void ByteUnitedMemoryLinks_Drop(void* self);
106
107     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
108     public static extern void UInt16UnitedMemoryLinks_Drop(void* self);
109
110     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
111     public static extern void UInt32UnitedMemoryLinks_Drop(void* self);
112
113     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
114     public static extern void UInt64UnitedMemoryLinks_Drop(void* self);
115
116     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
117     public static extern byte ByteUnitedMemoryLinks_Create(void* self, byte* substitution,
118         ↪ nuint substitutionLength, CreateCallback_UInt8 callback);
119
120     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
121     public static extern ushort UInt16UnitedMemoryLinks_Create(void* self, ushort*
122         ↪ substitution, nuint substitutionLength, CreateCallback_UInt16 callback);
123
124     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
125     public static extern uint UInt32UnitedMemoryLinks_Create(void* self, uint* substitution,
126         ↪ nuint substitutionLength, CreateCallback_UInt32 callback);
127
128     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
129     public static extern ulong UInt64UnitedMemoryLinks_Create(void* self, ulong*
130         ↪ substitution, nuint substitutionLength, CreateCallback_UInt64 callback);
131
132     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
133     public static extern byte ByteUnitedMemoryLinks_Count(void* self, byte* restriction,
134         ↪ nuint len);
135
136     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
137     public static extern ushort UInt16UnitedMemoryLinks_Count(void* self, ushort*
138         ↪ restriction, nuint len);
139
140     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
141     public static extern uint UInt32UnitedMemoryLinks_Count(void* self, uint* restriction,
142         ↪ nuint len);
143
144     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
145     public static extern ulong UInt64UnitedMemoryLinks_Count(void* self, ulong* restriction,
146         ↪ nuint len);
147
148     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
149     public static extern byte ByteUnitedMemoryLinks_Each(void* self, byte* restriction,
150         ↪ nuint len, EachCallback_UInt8 callback);
151
152     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
153     public static extern ushort UInt16UnitedMemoryLinks_Each(void* self, ushort*
154         ↪ restriction, nuint len, EachCallback_UInt16 callback);
155
156     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
157     public static extern uint UInt32UnitedMemoryLinks_Each(void* self, uint* restriction,
158         ↪ nuint len, EachCallback_UInt32 callback);
159
160     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
161     public static extern ulong UInt64UnitedMemoryLinks_Each(void* self, ulong* restriction,
162         ↪ nuint len, EachCallback_UInt64 callback);
163
164     [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]

```



```

152 public static extern byte ByteUnitedMemoryLinks_Update(void* self, byte* restriction,
    ↳ nuint restrictionLength, byte* substitution, nuint substitutionLength,
    ↳ UpdateCallback_UInt8 callback);
153
154 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
155 public static extern ushort UInt16UnitedMemoryLinks_Update(void* self, ushort*
    ↳ restriction, nuint restrictionLength, ushort* substitution, nuint
    ↳ substitutionLength, UpdateCallback_UInt16 callback);
156
157 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
158 public static extern uint UInt32UnitedMemoryLinks_Update(void* self, uint* restriction,
    ↳ nuint restrictionLength, uint* substitution, nuint substitutionLength,
    ↳ UpdateCallback_UInt32 callback);
159
160 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
161 public static extern ulong UInt64UnitedMemoryLinks_Update(void* self, ulong*
    ↳ restriction, nuint restrictionLength, ulong* substitution, nuint
    ↳ substitutionLength, UpdateCallback_UInt64 callback);
162
163 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
164 public static extern byte ByteUnitedMemoryLinks_Delete(void* self, byte* restriction,
    ↳ nuint restrictionLength, DeleteCallback_UInt8 callback);
165
166 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
167 public static extern ushort UInt16UnitedMemoryLinks_Delete(void* self, ushort*
    ↳ restriction, nuint len, DeleteCallback_UInt16 callback);
168
169 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
170 public static extern uint UInt32UnitedMemoryLinks_Delete(void* self, uint* restriction,
    ↳ nuint len, DeleteCallback_UInt32 callback);
171
172 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
173 public static extern ulong UInt64UnitedMemoryLinks_Delete(void* self, ulong*
    ↳ restriction, nuint len, DeleteCallback_UInt64 callback);
174 }
175
176 public class UnitedMemoryLinks<TLinkAddress> : DisposableBase, ILinks<TLinkAddress> where
    ↳ TLinkAddress : struct
177 {
178     private static readonly UncheckedConverter<byte, TLinkAddress> from_u8 =
    ↳ UncheckedConverter<byte, TLinkAddress>.Default;
179     private static readonly UncheckedConverter<ushort, TLinkAddress> from_u16 =
    ↳ UncheckedConverter<ushort, TLinkAddress>.Default;
180     private static readonly UncheckedConverter<uint, TLinkAddress> from_u32 =
    ↳ UncheckedConverter<uint, TLinkAddress>.Default;
181     private static readonly UncheckedConverter<ulong, TLinkAddress> from_u64 =
    ↳ UncheckedConverter<ulong, TLinkAddress>.Default;
182     private static readonly UncheckedConverter<TLinkAddress, ulong> from_t =
    ↳ UncheckedConverter<TLinkAddress, ulong>.Default;
183
184     public LinksConstants<TLinkAddress> Constants { get; }
185
186     private readonly unsafe void* _ptr;
187
188     public UnitedMemoryLinks(string path)
189     {
190         TLinkAddress t = default;
191         unsafe
192         {
193             _ptr = t switch
194             {
195                 byte => Methods.ByteUnitedMemoryLinks_New(path),
196                 ushort => Methods.UInt16UnitedMemoryLinks_New(path),
197                 uint => Methods.UInt32UnitedMemoryLinks_New(path),
198                 ulong => Methods.UInt64UnitedMemoryLinks_New(path),
199                 _ => throw new NotImplementedException()
200             };
201
202             // TODO: Update api
203             Constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
    ↳ true);
204         }
205     }
206
207     public TLinkAddress Count(IList<TLinkAddress>? restriction)
208     {
209         var restrictionLength = restriction?.Count ?? 0;
210         unsafe
211         {

```

```

212 TLinkAddress t = default;
213 switch (t)
214 {
215     case byte:
216     {
217         var restrictionArray = stackalloc byte[restrictionLength];
218         var byteRestrictionArray = (IList<byte>)restriction;
219         for (var i = 0; i < restrictionLength; i++)
220         {
221             restrictionArray[i] = byteRestrictionArray[i];
222         }
223         return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Count(_ptr,
224             ↪ restrictionArray, (nuint)restrictionLength));
225     }
226     case ushort:
227     {
228         var restrictionArray = stackalloc ushort[restrictionLength];
229         var ushortRestrictionArray = (IList<ushort>)restriction;
230         for (var i = 0; i < restrictionLength; i++)
231         {
232             restrictionArray[i] = ushortRestrictionArray[i];
233         }
234         return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Count(_ptr,
235             ↪ restrictionArray, (nuint)restrictionLength));
236     }
237     case uint:
238     {
239         var restrictionArray = stackalloc uint[restrictionLength];
240         var uintRestrictionArray = (IList<uint>)restriction;
241         for (var i = 0; i < restrictionLength; i++)
242         {
243             restrictionArray[i] = uintRestrictionArray[i];
244         }
245         return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Count(_ptr,
246             ↪ restrictionArray, (nuint)restrictionLength));
247     }
248     case ulong:
249     {
250         {
251             var restrictionArray = stackalloc ulong[restrictionLength];
252             var ulongRestrictionArray = (IList<ulong>)restriction;
253             for (var i = 0; i < restrictionLength; i++)
254             {
255                 restrictionArray[i] = ulongRestrictionArray[i];
256             }
257             return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Count(_ptr,
258                 ↪ restrictionArray, (nuint)restrictionLength));
259         }
260     }
261     default:
262     {
263         throw new NotImplementedException();
264     }
265 }
266 }
267 }
268
269 public TLinkAddress Each(IList<TLinkAddress>? restriction, ReadHandler<TLinkAddress>?
270     ↪ handler)
271 {
272     var restrictionLength = restriction?.Count ?? 0;
273     unsafe
274     {
275         TLinkAddress t = default;
276         switch (t)
277         {
278             case byte:
279             {
280                 byte Callback(FfiLink_UInt8 link) =>
281                 ↪ (byte)from_t.Convert(handler?.Invoke(new
282                 ↪ Link<TLinkAddress>(from_u8.Convert(link.Index),
283                 ↪ from_u8.Convert(link.Source), from_u8.Convert(link.Target))) ??
284                 ↪ Constants.Continue);
285                 var restrictionArray = stackalloc byte[restrictionLength];
286                 var byteRestrictionArray = (IList<byte>)restriction;
287                 for (var i = 0; i < restrictionLength; i++)
288                 {
289                     restrictionArray[i] = byteRestrictionArray[i];
290                 }
291             }
292         }
293     }
294 }

```

```

281     }
282     return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Each(_ptr,
    ↪ restrictionArray, (nuint)restrictionLength, Callback));
283 }
284 case ushort:
285 {
286     ushort Callback(FfiLink_UInt16 link) =>
    ↪ (ushort)from_t.Convert(handler?.Invoke(new
    ↪ Link<TLinkAddress>(from_u16.Convert(link.Index),
    ↪ from_u16.Convert(link.Source), from_u16.Convert(link.Target))) ??
    ↪ Constants.Continue);
287     var restrictionArray = stackalloc ushort[restrictionLength];
288     var ushortRestrictionArray = (IList<ushort>)restriction;
289     for (var i = 0; i < restrictionLength; i++)
290     {
291         restrictionArray[i] = ushortRestrictionArray[i];
292     }
293     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Each(_ptr,
    ↪ restrictionArray, (nuint)restrictionLength, Callback));
294 }
295 case uint:
296 {
297     uint Callback(FfiLink_UInt32 link) =>
    ↪ (uint)from_t.Convert(handler?.Invoke(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?.Invoke(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) =>
338         (byte)from_t.Convert(handler?.Invoke(new
339             ↳ Link<TLinkAddress>(from_u8.Convert(before.Index),
340             ↳ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
341             ↳ Link<TLinkAddress>(from_u8.Convert(after.Index),
342             ↳ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) ??
343             ↳ Constants.Continue);
344     var substitutionArray = stackalloc byte[substitutionLength];
345     var byteSubstitutionArray = (IList<byte>)substitution;
346     for (var i = 0; i < substitutionLength; i++)
347     {
348         substitutionArray[i] = byteSubstitutionArray[i];
349     }
350     return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Create(_ptr,
351         ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
352 }
353 case ushort:
354 {
355     ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
356         (ushort)from_t.Convert(handler?.Invoke(new
357             ↳ Link<TLinkAddress>(from_u16.Convert(before.Index),
358             ↳ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
359             ↳ new Link<TLinkAddress>(from_u16.Convert(after.Index),
360             ↳ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) ??
361             ↳ Constants.Continue);
362     var substitutionArray = stackalloc ushort[substitutionLength];
363     var ushortSubstitutionArray = (IList<ushort>)substitution;
364     for (var i = 0; i < substitutionLength; i++)
365     {
366         substitutionArray[i] = ushortSubstitutionArray[i];
367     }
368     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Create(_ptr,
369         ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
370 }
371 case uint:
372 {
373     uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
374         (uint)from_t.Convert(handler?.Invoke(new
375             ↳ Link<TLinkAddress>(from_u32.Convert(before.Index),
376             ↳ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
377             ↳ new Link<TLinkAddress>(from_u32.Convert(after.Index),
378             ↳ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) ??
379             ↳ Constants.Continue);
380     var substitutionArray = stackalloc uint[substitutionLength];
381     var uintSubstitutionArray = (IList<uint>)substitution;
382     for (var i = 0; i < substitutionLength; i++)
383     {
384         substitutionArray[i] = uintSubstitutionArray[i];
385     }
386     return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Create(_ptr,
387         ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
388 }
389 case ulong:
390 {
391     ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
392         (ulong)from_t.Convert(handler?.Invoke(new
393             ↳ Link<TLinkAddress>(from_u64.Convert(before.Index),
394             ↳ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
395             ↳ new Link<TLinkAddress>(from_u64.Convert(after.Index),
396             ↳ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) ??
397             ↳ Constants.Continue);
398     var substitutionArray = stackalloc ulong[substitutionLength];
399     var ulongSubstitutionArray = (IList<ulong>)substitution;
400     for (var i = 0; i < substitutionLength; i++)
401     {
402         substitutionArray[i] = ulongSubstitutionArray[i];
403     }
404     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Create(_ptr,
405         ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
406 }
407 default:
408 {
409     throw new NotImplementedException();
410 }
411 };
412 }

```

```

387 }
388
389 public TLinkAddress Update(IList<TLinkAddress>? restriction, IList<TLinkAddress>?
→ substitution, WriteHandler<TLinkAddress>? handler)
390 {
391     var restrictionLength = restriction?.Count ?? 0;
392     var substitutionLength = substitution?.Count ?? 0;
393     unsafe
394     {
395         TLinkAddress t = default;
396         switch (t)
397         {
398             case byte:
399                 {
400                     var restrictionArray = stackalloc byte[restrictionLength];
401                     var byteRestrictionArray = (IList<byte>)restriction;
402                     for (var i = 0; i < restrictionLength; i++)
403                     {
404                         restrictionArray[i] = byteRestrictionArray[i];
405                     }
406                     var substitutionArray = stackalloc byte[substitutionLength];
407                     var byteSubstitutionArray = (IList<byte>)substitution;
408                     for (var i = 0; i < substitutionLength; i++)
409                     {
410                         substitutionArray[i] = byteSubstitutionArray[i];
411                     }
412                     byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
→ (byte)from_t.Convert(handler?.Invoke(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);
413                     return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Update(_ptr,
→ restrictionArray, (nuint)restrictionLength, substitutionArray,
→ (nuint)(substitution?.Count ?? 0), Callback));
414                 }
415             case ushort:
416                 {
417                     var restrictionArray = stackalloc ushort[restrictionLength];
418                     var ushortRestrictionArray = (IList<ushort>)restriction;
419                     for (var i = 0; i < restrictionLength; i++)
420                     {
421                         restrictionArray[i] = ushortRestrictionArray[i];
422                     }
423                     var substitutionArray = stackalloc ushort[substitutionLength];
424                     var ushortSubstitutionArray = (IList<ushort>)substitution;
425                     for (var i = 0; i < substitutionLength; i++)
426                     {
427                         substitutionArray[i] = ushortSubstitutionArray[i];
428                     }
429                     ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
→ (ushort)from_t.Convert(handler?.Invoke(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);
430                     return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Update(_ptr,
→ restrictionArray, (nuint)restrictionLength, substitutionArray,
→ (nuint)(substitution?.Count ?? 0), Callback));
431                 }
432             case uint:
433                 {
434                     var restrictionArray = stackalloc uint[restrictionLength];
435                     var uintRestrictionArray = (IList<uint>)restriction;
436                     for (var i = 0; i < restrictionLength; i++)
437                     {
438                         restrictionArray[i] = uintRestrictionArray[i];
439                     }
440                     var substitutionArray = stackalloc uint[substitutionLength];
441                     var uintSubstitutionArray = (IList<uint>)substitution;
442                     for (var i = 0; i < substitutionLength; i++)
443                     {
444                         substitutionArray[i] = uintSubstitutionArray[i];
445                     }

```

```

446         uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
447         ↪ (uint)from_t.Convert(handler?.Invoke(new
448         ↪ Link<TLinkAddress>(from_u32.Convert(before.Index),
449         ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
450         ↪ new Link<TLinkAddress>(from_u32.Convert(after.Index),
451         ↪ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) ??
452         ↪ Constants.Continue);
453     return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Update(_ptr,
454     ↪ restrictionArray, (nuint)restrictionLength, substitutionArray,
455     ↪ (nuint)(substitution?.Count ?? 0), Callback));
456 }
457 case ulong:
458 {
459     var restrictionArray = stackalloc ulong[restrictionLength];
460     var ulongRestrictionArray = (IList<ulong>)restriction;
461     for (var i = 0; i < restrictionLength; i++)
462     {
463         restrictionArray[i] = ulongRestrictionArray[i];
464     }
465     var substitutionArray = stackalloc ulong[substitutionLength];
466     var ulongSubstitutionArray = (IList<ulong>)substitution;
467     for (var i = 0; i < substitutionLength; i++)
468     {
469         substitutionArray[i] = ulongSubstitutionArray[i];
470     }
471     ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
472     ↪ (ulong)from_t.Convert(handler?.Invoke(new
473     ↪ Link<TLinkAddress>(from_u64.Convert(before.Index),
474     ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
475     ↪ new Link<TLinkAddress>(from_u64.Convert(after.Index),
476     ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) ??
477     ↪ Constants.Continue);
478     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Update(_ptr,
479     ↪ restrictionArray, (nuint)restrictionLength, substitutionArray,
480     ↪ (nuint)(substitution?.Count ?? 0), Callback));
481 }
482 default:
483 {
484     throw new NotImplementedException();
485 }
486 }
487 }
488 }
489 }
490
491 public TLinkAddress Delete(IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
492 ↪ handler)
493 {
494     var restrictionLength = restriction?.Count ?? 0;
495     unsafe
496     {
497         TLinkAddress t = default;
498         switch (t)
499         {
500             case byte:
501             {
502                 var restrictionArray = stackalloc byte[restrictionLength];
503                 var byteRestrictionArray = (IList<byte>)restriction;
504                 for (var i = 0; i < restrictionLength; i++)
505                 {
506                     restrictionArray[i] = byteRestrictionArray[i];
507                 }
508                 byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
509                 ↪ (byte)from_t.Convert(handler?.Invoke(new
510                 ↪ Link<TLinkAddress>(from_u8.Convert(before.Index),
511                 ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
512                 ↪ Link<TLinkAddress>(from_u8.Convert(after.Index),
513                 ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) ??
514                 ↪ Constants.Continue);
515                 return (TLinkAddress)(object)Methods.ByteUnitedMemoryLinks_Delete(_ptr,
516                 ↪ restrictionArray, (nuint)restrictionLength, Callback);
517             }
518             case ushort:
519             {
520                 var restrictionArray = stackalloc ushort[restrictionLength];
521                 var ushortRestrictionArray = (IList<ushort>)restriction;
522                 for (var i = 0; i < restrictionLength; i++)
523                 {
524                     restrictionArray[i] = ushortRestrictionArray[i];
525                 }
526             }
527         }
528     }
529 }

```

```

500     }
501     ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
        (ushort)from_t.Convert(handler?.Invoke(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);
502     return
        (TLinkAddress)(object)Methods.UInt16UnitedMemoryLinks_Delete(_ptr,
        restrictionArray, (nuint)restrictionLength, Callback);
503 }
504 case uint:
505 {
506     var restrictionArray = stackalloc uint[restrictionLength];
507     var uintRestrictionArray = (IList<uint>)restriction;
508     for (var i = 0; i < restrictionLength; i++)
509     {
510         restrictionArray[i] = uintRestrictionArray[i];
511     }
512     uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
        (uint)from_t.Convert(handler?.Invoke(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?.Invoke(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         {

```



```

49     var random = RandomHelpers.Default;
50     var addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
51     var uint64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
52     for (var i = 0UL; i < amountOfCreations; i++)
53     {
54         var linksAddressRange = new Range<ulong>(0,
55             ↪ addressToUInt64Converter.Convert(links.Count()));
56         var source =
57             ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
58         var target =
59             ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
60         links.GetOrCreate(source, target);
61     }
62 }
63
64 /// <summary>
65 /// <para>
66 /// Runs the random searches using the specified links.
67 /// </para>
68 /// <para></para>
69 /// </summary>
70 /// <typeparam name="TLinkAddress">
71 /// <para>The link.</para>
72 /// <para></para>
73 /// </typeparam>
74 /// <param name="links">
75 /// <para>The links.</para>
76 /// <para></para>
77 /// </param>
78 /// <param name="amountOfSearches">
79 /// <para>The amount of searches.</para>
80 /// <para></para>
81 /// </param>
82 [MethodImpl(MethodImplOptions.AggressiveInlining)]
83 public static void RunRandomSearches<TLinkAddress>(this ILinks<TLinkAddress> links,
84     ↪ ulong amountOfSearches)
85 {
86     var random = RandomHelpers.Default;
87     var addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
88     var uint64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
89     for (var i = 0UL; i < amountOfSearches; i++)
90     {
91         var linksAddressRange = new Range<ulong>(0,
92             ↪ addressToUInt64Converter.Convert(links.Count()));
93         var source =
94             ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
95         var target =
96             ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
97         links.SearchOrDefault(source, target);
98     }
99 }
100
101 /// <summary>
102 /// <para>
103 /// Runs the random deletions using the specified links.
104 /// </para>
105 /// <para></para>
106 /// </summary>
107 /// <typeparam name="TLinkAddress">
108 /// <para>The link.</para>
109 /// <para></para>
110 /// </typeparam>
111 /// <param name="links">
112 /// <para>The links.</para>
113 /// <para></para>
114 /// </param>
115 /// <param name="amountOfDeletions">
116 /// <para>The amount of deletions.</para>
117 /// <para></para>
118 /// </param>
119 [MethodImpl(MethodImplOptions.AggressiveInlining)]
120 public static void RunRandomDeletions<TLinkAddress>(this ILinks<TLinkAddress> links,
121     ↪ ulong amountOfDeletions)
122 {
123     var random = RandomHelpers.Default;
124     var addressToUInt64Converter = UncheckedConverter<TLinkAddress, ulong>.Default;
125     var uint64ToAddressConverter = UncheckedConverter<ulong, TLinkAddress>.Default;
126     var linksCount = addressToUInt64Converter.Convert(links.Count());

```

```

119     var min = amountOfDeletions > linksCount ? OUL : linksCount - amountOfDeletions;
120     for (var i = OUL; i < amountOfDeletions; i++)
121     {
122         linksCount = addressToUInt64Converter.Convert(links.Count());
123         if (linksCount <= min)
124         {
125             break;
126         }
127         var linksAddressRange = new Range<ulong>(min, linksCount);
128         var link =
129             ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
130         links.Delete(link);
131     }
132
133     /// <summary>
134     /// <para>
135     /// Deletes the links.
136     /// </para>
137     /// <para></para>
138     /// </summary>
139     /// <typeparam name="TLinkAddress">
140     /// <para>The link.</para>
141     /// <para></para>
142     /// </typeparam>
143     /// <param name="links">
144     /// <para>The links.</para>
145     /// <para></para>
146     /// </param>
147     /// <param name="linkToDelete">
148     /// <para>The link to delete.</para>
149     /// <para></para>
150     /// </param>
151     [MethodImpl(MethodImplOptions.AggressiveInlining)]
152     public static TLinkAddress Delete<TLinkAddress>(this ILinks<TLinkAddress> links,
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
162     /// <remarks>
163     /// TODO: Возможно есть очень простой способ это сделать.
164     /// (Например просто удалить файл, или изменить его размер таким образом,
165     /// чтобы удалился весь контент)
166     /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
167     /// </remarks>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     public static void DeleteAll<TLinkAddress>(this ILinks<TLinkAddress> links)
170     {
171         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
172         var comparer = Comparer<TLinkAddress>.Default;
173         for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
174             ↪ Arithmetic.Decrement(i))
175         {
176             links.Delete(i);
177             if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
178             {
179                 i = links.Count();
180             }
181         }
182     }
183
184     /// <summary>
185     /// <para>
186     /// Firsts the links.
187     /// </para>
188     /// <para></para>
189     /// </summary>
190     /// <typeparam name="TLinkAddress">
191     /// <para>The link.</para>
192     /// <para></para>
193     /// </typeparam>
194     /// <param name="links">
195     /// <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
225             ↳ найдено связей.");
226     }
227     return firstLink;
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
253     ↳ ILinks<TLinkAddress> links, IList<TLinkAddress>? query)
254 {
255     IList<TLinkAddress>? result = null;
256     var count = 0;
257     var constants = links.Constants;
258     var @continue = constants.Continue;
259     var @break = constants.Break;
260     links.Each(query, linkHandler);
261     return result;
262
263     TLinkAddress linkHandler(IList<TLinkAddress>? link)
264     {
265         if (count == 0)
266         {
267             result = link;
268             count++;
269             return @continue;
270         }
271         else

```

```

271     {
272         result = null;
273         return @break;
274     }
275 }
276
277 #region Paths
278
279 /// <remarks>
280 /// TODO: Как так? Как то что ниже может быть корректно?
281 /// Скорее всего практически не применимо
282 /// Предполагалось, что можно было конвертировать формируемый в проходе через
283     ↪ SequenceWalker
284 /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
285 /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
286 /// </remarks>
287 [MethodImpl(MethodImplOptions.AggressiveInlining)]
288 public static bool CheckPathExistence<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 = values[constants.SourcePart];
304         var target = values[constants.TargetPart];
305         if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
306             ↪ next))
307         {
308             //throw new InvalidOperationException(string.Format("Невозможно выбрать
309             ↪ путь, так как и Source и Target совпадают с элементом пути {0}.", next));
310             return false;
311         }
312         if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
313             ↪ target))
314         {
315             //throw new InvalidOperationException(string.Format("Невозможно продолжить
316             ↪ путь через элемент пути {0}", next));
317             return false;
318         }
319         current = next;
320     }
321     return true;
322 }
323
324 /// <remarks>
325 /// Может потребовать дополнительного стека для PathElement's при использовании
326     ↪ SequenceWalker.
327 /// </remarks>
328 [MethodImpl(MethodImplOptions.AggressiveInlining)]
329 public static TLinkAddress GetByKey<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 /// </summary>
346 /// <typeparam name="TLinkAddress">

```

```

341 /// <para>The link.</para>
342 /// <para></para>
343 /// </typeparam>
344 /// <param name="links">
345 /// <para>The links.</para>
346 /// <para></para>
347 /// </param>
348 /// <param name="root">
349 /// <para>The root.</para>
350 /// <para></para>
351 /// </param>
352 /// <param name="size">
353 /// <para>The size.</para>
354 /// <para></para>
355 /// </param>
356 /// <param name="index">
357 /// <para>The index.</para>
358 /// <para></para>
359 /// </param>
360 /// <exception cref="ArgumentOutOfRangeException">
361 /// <para>Sequences with sizes other than powers of two are not supported.</para>
362 /// <para></para>
363 /// </exception>
364 /// <returns>
365 /// <para>The current link.</para>
366 /// <para></para>
367 /// </returns>
368 [MethodImpl(MethodImplOptions.AggressiveInlining)]
369 public static TLinkAddress GetSquareMatrixSequenceElementByIndex<TLinkAddress>(this
    ↪ ILinks<TLinkAddress> links, TLinkAddress root, ulong size, ulong index)
370 {
371     var constants = links.Constants;
372     var source = constants.SourcePart;
373     var target = constants.TargetPart;
374     if (!Platform.Numbers.Math.IsPowerOfTwo(size))
375     {
376         throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
            ↪ than powers of two are not supported.");
377     }
378     var path = new BitArray(BitConverter.GetBytes(index));
379     var length = Bit.GetLowestPosition(size);
380     links.EnsureLinkExists(root, "root");
381     var currentLink = root;
382     for (var i = length - 1; i >= 0; i--)
383     {
384         currentLink = links.GetLink(currentLink)[path[i] ? target : source];
385     }
386     return currentLink;
387 }
388
389 #endregion
390
391 /// <summary>
392 /// Возвращает индекс указанной связи.
393 /// </summary>
394 /// <param name="links">Хранилище связей.</param>
395 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↪ содержимого.</param>
396 /// <returns>Индекс начальной связи для указанной связи.</returns>
397 [MethodImpl(MethodImplOptions.AggressiveInlining)]
398 public static TLinkAddress GetIndex<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ IList<TLinkAddress>? link) => link[links.Constants.IndexPart];
399
400 /// <summary>
401 /// Возвращает индекс начальной (Source) связи для указанной связи.
402 /// </summary>
403 /// <param name="links">Хранилище связей.</param>
404 /// <param name="link">Индекс связи.</param>
405 /// <returns>Индекс начальной связи для указанной связи.</returns>
406 [MethodImpl(MethodImplOptions.AggressiveInlining)]
407 public static TLinkAddress GetSource<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ TLinkAddress link) => links.GetLink(link)[links.Constants.SourcePart];
408
409 /// <summary>
410 /// Возвращает индекс начальной (Source) связи для указанной связи.
411 /// </summary>
412 /// <param name="links">Хранилище связей.</param>

```

```

413 /// <param name="link">Связь представленная списком, состоящим из её адреса и
414   ↳ содержимого.</param>
415 /// <returns>Индекс начальной связи для указанной связи.</returns>
416 [MethodImpl(MethodImplOptions.AggressiveInlining)]
417 public static TLinkAddress GetSource<TLinkAddress>(this ILinks<TLinkAddress> links,
418   ↳ IList<TLinkAddress>? link) => link[links.Constants.SourcePart];
419
420 /// <summary>
421 /// Возвращает индекс конечной (Target) связи для указанной связи.
422 /// </summary>
423 /// <param name="links">Хранилище связей.</param>
424 /// <param name="link">Индекс связи.</param>
425 /// <returns>Индекс конечной связи для указанной связи.</returns>
426 [MethodImpl(MethodImplOptions.AggressiveInlining)]
427 public static TLinkAddress GetTarget<TLinkAddress>(this ILinks<TLinkAddress> links,
428   ↳ TLinkAddress link) => links.GetLink(link)[links.Constants.TargetPart];
429
430 /// <summary>
431 /// Возвращает индекс конечной (Target) связи для указанной связи.
432 /// </summary>
433 /// <param name="links">Хранилище связей.</param>
434 /// <param name="link">Связь представленная списком, состоящим из её адреса и
435   ↳ содержимого.</param>
436 /// <returns>Индекс конечной связи для указанной связи.</returns>
437 [MethodImpl(MethodImplOptions.AggressiveInlining)]
438 public static TLinkAddress GetTarget<TLinkAddress>(this ILinks<TLinkAddress> links,
439   ↳ IList<TLinkAddress>? link) => link[links.Constants.TargetPart];
440
441 /// <summary>
442 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
443   ↳ (handler) для каждой подходящей связи.
444 /// </summary>
445 /// <param name="links">Хранилище связей.</param>
446 /// <param name="handler">Обработчик каждой подходящей связи.</param>
447 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
448   ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Any -
449   ↳ отсутствие ограничения, 1..∞ конкретный адрес связи.</param>
450 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
451   ↳ случае.</returns>
452 [MethodImpl(MethodImplOptions.AggressiveInlining)]
453 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
454   ↳ ReadHandler<TLinkAddress>? handler, params TLinkAddress[] restriction)
455   => EqualityComparer<TLinkAddress>.Default.Equals(links.Each(restriction, handler),
456     ↳ links.Constants.Continue);
457
458 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
459   ↳ Func<TLinkAddress, bool> handler, TLinkAddress source, TLinkAddress target) =>
460   ↳ links.Each(source, target, handler);
461
462 /// <summary>
463 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
464   ↳ (handler) для каждой подходящей связи.
465 /// </summary>
466 /// <param name="links">Хранилище связей.</param>
467 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
468   ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
469   ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
470 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
471   ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
472   ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
473 /// <param name="handler">Обработчик каждой подходящей связи.</param>
474 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
475   ↳ случае.</returns>
476 [MethodImpl(MethodImplOptions.AggressiveInlining)]
477 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
478   ↳ source, TLinkAddress target, Func<TLinkAddress, bool> handler)
479 {
480     var constants = links.Constants;
481     return links.Each(link => handler(link[constants.IndexPart]) ? constants.Continue :
482       ↳ constants.Break, constants.Any, source, target);
483 }
484
485 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links,
486   ↳ ReadHandler<TLinkAddress>? handler, TLinkAddress source, TLinkAddress target) =>
487   ↳ links.Each(source, target, handler);

```

```

466 /// <summary>
467 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
468   ↳ (handler) для каждой подходящей связи.
469 /// </summary>
470 /// <param name="links">Хранилище связей.</param>
471 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
472   ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
473   ↳ Constants.Any - любое начало, 1..∞ конкретное начало)</param>
474 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
475   ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
476   ↳ Constants.Any - любой конец, 1..∞ конкретный конец)</param>
477 /// <param name="handler">Обработчик каждой подходящей связи.</param>
478 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
479   ↳ случае.</returns>
480 [MethodImpl(MethodImplOptions.AggressiveInlining)]
481 public static bool Each<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
482   ↳ source, TLinkAddress target, ReadHandler<TLinkAddress>? handler) =>
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)]

```

```

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

```



```

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

```

```

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

```

```

750 /// <para></para>
751 /// </exception>
752 [MethodImpl(MethodImplOptions.AggressiveInlining)]
753 public static void EnsureLinkIsItselfOrExists<TLinkAddress>(this ILinks<TLinkAddress>
    ↪ links, TLinkAddress link, string argumentName)
754 {
755     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
756     if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))
757     {
758         throw new ArgumentLinkDoesNotExistsException<TLinkAddress>(link, argumentName);
759     }
760 }
761
762 /// <param name="links">Хранилище связей.</param>
763 [MethodImpl(MethodImplOptions.AggressiveInlining)]
764 public static void EnsureDoesNotExists<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ TLinkAddress source, TLinkAddress target)
765 {
766     if (links.Exists(source, target))
767     {
768         throw new LinkWithSameValueAlreadyExistsException();
769     }
770 }
771
772 /// <param name="links">Хранилище связей.</param>
773 [MethodImpl(MethodImplOptions.AggressiveInlining)]
774 public static void EnsureNoUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ TLinkAddress link)
775 {
776     if (links.HasUsages(link))
777     {
778         throw new ArgumentLinkHasDependenciesException<TLinkAddress>(link);
779     }
780 }
781
782 /// <param name="links">Хранилище связей.</param>
783 [MethodImpl(MethodImplOptions.AggressiveInlining)]
784 public static void EnsureCreated<TLinkAddress>(this ILinks<TLinkAddress> links, params
    ↪ TLinkAddress[] addresses) => links.EnsureCreated(links.Create, addresses);
785
786 /// <param name="links">Хранилище связей.</param>
787 [MethodImpl(MethodImplOptions.AggressiveInlining)]
788 public static void EnsurePointsCreated<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ params TLinkAddress[] addresses) => links.EnsureCreated(links.CreatePoint,
    ↪ addresses);
789
790 /// <param name="links">Хранилище связей.</param>
791 [MethodImpl(MethodImplOptions.AggressiveInlining)]
792 public static void EnsureCreated<TLinkAddress>(this ILinks<TLinkAddress> links,
    ↪ Func<TLinkAddress> creator, params TLinkAddress[] addresses)
793 {
794     var addressToUInt64Converter = CheckedConverter<TLinkAddress, ulong>.Default;
795     var uInt64ToAddressConverter = CheckedConverter<ulong, TLinkAddress>.Default;
796     var nonExistentAddresses = new HashSet<TLinkAddress>(addresses.Where(x =>
    ↪ !links.Exists(x)));
797     if (nonExistentAddresses.Count > 0)
798     {
799         var max = nonExistentAddresses.Max();
800         max = uInt64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
    ↪ Convert(max),
    ↪ addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
    ↪ imum)));
801         var createdLinks = new List<TLinkAddress>();
802         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
803         TLinkAddress createdLink = creator();
804         while (!equalityComparer.Equals(createdLink, max))
805         {
806             createdLinks.Add(createdLink);
807         }
808         for (var i = 0; i < createdLinks.Count; i++)
809         {
810             if (!nonExistentAddresses.Contains(createdLinks[i]))
811             {
812                 links.Delete(createdLinks[i]);
813             }
814         }
815     }

```

```

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

```

```

881 {
882     var constants = links.Constants;
883     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
884         ↪ constants.Break, handler);
885     TLinkAddress link = default;
886     TLinkAddress HandlerWrapper(IList<TLinkAddress>? before, IList<TLinkAddress>? after)
887     {
888         link = after[constants.IndexPart];
889         return handlerState.Handler != null ? handlerState.Handler(before, after) :
890             ↪ constants.Continue;
891     }
892     handlerState.Apply(links.Create(null, HandlerWrapper));
893     handlerState.Apply(links.Update(link, link, link, HandlerWrapper));
894     return handlerState.Result;
895 }
896
897 public static TLinkAddress CreateAndUpdate<TLinkAddress>(this ILinks<TLinkAddress>
898     ↪ links, TLinkAddress source, TLinkAddress target)
899 {
900     var constants = links.Constants;
901     var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
902         ↪ constants.Break);
903     links.CreateAndUpdate(source, target, setter.SetFirstFromSecondListAndReturnTrue);
904     return setter.Result;
905 }
906
907 /// <param name="links">Хранилище связей.</param>
908 [MethodImpl(MethodImplOptions.AggressiveInlining)]
909 public static TLinkAddress CreateAndUpdate<TLinkAddress>(this ILinks<TLinkAddress>
910     ↪ links, TLinkAddress source, TLinkAddress target, WriteHandler<TLinkAddress>? handler)
911 {
912     var constants = links.Constants;
913     TLinkAddress createdLink = default;
914     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
915         ↪ constants.Break, handler);
916     handlerState.Apply(links.Create(null, (before, after) =>
917     {
918         createdLink = links.GetIndex(after);
919         return handlerState.Handler != null ? handlerState.Handler(before, after) :
920             ↪ constants.Continue;
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 }

```

```

}

/// <summary>
/// Обновляет связь с указанными началом (Source) и концом (Target)
/// на связь с указанными началом (NewSource) и концом (NewTarget).
/// </summary>
/// <param name="links">Хранилище связей.</param>
/// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
→ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Itself -
→ требование установить ссылку на себя, 1.. $\infty$  конкретный адрес другой связи.</param>
/// <returns>Индекс обновлённой связи.</returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static TLinkAddress Update<TLinkAddress>(this ILinks<TLinkAddress> links,
→ IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>? handler)
{
    return restriction.Count switch
    {
        2 => links.MergeAndDelete(restriction[0], restriction[1], handler),
        4 => links.UpdateOrCreateOrGet(restriction[0], restriction[1], restriction[2],
→ restriction[3], handler),
        _ => links.Update(restriction[0], restriction[1], restriction[2], handler)
    };
}

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);

/// <summary>
/// <para>
/// Resolves the constant as self reference using the specified links.
/// </para>
/// <para></para>
/// </summary>
/// <typeparam name="TLinkAddress">
/// <para>The link.</para>
/// <para></para>
/// </typeparam>
/// <param name="links">
/// <para>The links.</para>
/// <para></para>
/// </param>
/// <param name="constant">
/// <para>The constant.</para>
/// <para></para>
/// </param>
/// <param name="restriction">
/// <para>The restriction.</para>
/// <para></para>
/// </param>
/// <param name="substitution">
/// <para>The substitution.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>A list of t link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static IList<TLinkAddress>? ResolveConstantAsSelfReference<TLinkAddress>(this
→ ILinks<TLinkAddress> links, TLinkAddress constant, IList<TLinkAddress>? restriction,
→ IList<TLinkAddress>? substitution)
{
    var equalityComparer = EqualityComparer<TLinkAddress>.Default;
    var constants = links.Constants;
    var restrictionIndex = restriction[constants.IndexPart];
    var substitutionIndex = substitution[constants.IndexPart];
    if (equalityComparer.Equals(substitutionIndex, default))
    {
        substitutionIndex = restrictionIndex;
    }
    var source = substitution[constants.SourcePart];
    var target = substitution[constants.TargetPart];
    source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;
    target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
}

```

```

1010         return new Link<TLinkAddress>(substitutionIndex, source, target);
1011     }
1012
1013     /// <summary>
1014     /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
1015     /// с указанными Source (началом) и Target (концом).
1016     /// </summary>
1017     /// <param name="links">Хранилище связей.</param>
1018     /// <param name="source">Индекс связи, которая является началом на создаваемой
1019     /// связи.</param>
1020     /// <param name="target">Индекс связи, которая является концом для создаваемой
1021     /// связи.</param>
1022     /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
1023     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1024     public static TLinkAddress GetOrCreate<TLinkAddress>(this ILinks<TLinkAddress> links,
1025     ↪ TLinkAddress source, TLinkAddress target)
1026     {
1027         var link = links.SearchOrDefault(source, target);
1028         if (EqualityComparer<TLinkAddress>.Default.Equals(link, default))
1029         {
1030             link = links.CreateAndUpdate(source, target);
1031         }
1032         return link;
1033     }
1034
1035     public static TLinkAddress UpdateOrCreateOrGet<TLinkAddress>(this ILinks<TLinkAddress>
1036     ↪ links, TLinkAddress source, TLinkAddress target, TLinkAddress newSource,
1037     ↪ TLinkAddress newTarget)
1038     {
1039         var constants = links.Constants;
1040         var setter = new Setter<TLinkAddress, TLinkAddress>(constants.Continue,
1041         ↪ constants.Break);
1042         links.UpdateOrCreateOrGet(source, target, newSource, newTarget,
1043         ↪ setter.SetFirstFromSecondListAndReturnTrue);
1044         return setter.Result;
1045     }
1046
1047     /// <summary>
1048     /// Обновляет связь с указанными началом (Source) и концом (Target)
1049     /// на связь с указанными началом (NewSource) и концом (NewTarget).
1050     /// </summary>
1051     /// <param name="links">Хранилище связей.</param>
1052     /// <param name="source">Индекс связи, которая является началом обновляемой
1053     /// связи.</param>
1054     /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
1055     /// <param name="newSource">Индекс связи, которая является началом связи, на которую
1056     /// выполняется обновление.</param>
1057     /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
1058     /// выполняется обновление.</param>
1059     /// <returns>Индекс обновлённой связи.</returns>
1060     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1061     public static TLinkAddress UpdateOrCreateOrGet<TLinkAddress>(this ILinks<TLinkAddress>
1062     ↪ links, TLinkAddress source, TLinkAddress target, TLinkAddress newSource,
1063     ↪ TLinkAddress newTarget, WriteHandler<TLinkAddress>? handler)
1064     {
1065         var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1066         var link = links.SearchOrDefault(source, target);
1067         if (equalityComparer.Equals(link, default))
1068         {
1069             return links.CreateAndUpdate(newSource, newTarget, handler);
1070         }
1071         if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
1072         ↪ target))
1073         {
1074             var linkStruct = new Link<TLinkAddress>(link, source, target);
1075             return link;
1076         }
1077         return links.Update(link, newSource, newTarget, handler);
1078     }
1079
1080     /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1081     /// <param name="links">Хранилище связей.</param>
1082     /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
1083     /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
1084     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1085     public static TLinkAddress DeleteIfExists<TLinkAddress>(this ILinks<TLinkAddress> links,
1086     ↪ TLinkAddress source, TLinkAddress target)

```

```

1072 {
1073     var link = links.SearchOrDefault(source, target);
1074     if (!EqualityComparer<TLinkAddress>.Default.Equals(link, default))
1075     {
1076         links.Delete(link);
1077         return link;
1078     }
1079     return default;
1080 }
1081
1082 /// <summary>Удаляет несколько связей.</summary>
1083 /// <param name="links">Хранилище связей.</param>
1084 /// <param name="deletedLinks">Список адресов связей к удалению.</param>
1085 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1086 public static void DeleteMany<TLinkAddress>(this ILinks<TLinkAddress> links,
1087     ↪ IList<TLinkAddress>? deletedLinks)
1088 {
1089     for (int i = 0; i < deletedLinks.Count; i++)
1090     {
1091         links.Delete(deletedLinks[i]);
1092     }
1093 }
1094
1095 public static void DeleteAllUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
1096     ↪ TLinkAddress linkIndex) => links.DeleteAllUsages(linkIndex, null);
1097
1098 /// <remarks>Before execution of this method ensure that deleted link is detached (all
1099     ↪ values - source and target are reset to null) or it might enter into infinite
1100     ↪ recursion.</remarks>
1101 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1102 public static TLinkAddress DeleteAllUsages<TLinkAddress>(this ILinks<TLinkAddress>
1103     ↪ links, TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1104 {
1105     var constants = links.Constants;
1106     var any = constants.Any;
1107     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1108     var usagesAsSourceQuery = new Link<TLinkAddress>(any, linkIndex, any);
1109     var usagesAsTargetQuery = new Link<TLinkAddress>(any, any, linkIndex);
1110     var usages = new List<IList<TLinkAddress>?>();
1111     var usagesFiller = new ListFiller<IList<TLinkAddress>?, TLinkAddress>(usages,
1112     ↪ constants.Continue);
1113     links.Each(usagesFiller.AddAndReturnConstant, usagesAsSourceQuery);
1114     links.Each(usagesFiller.AddAndReturnConstant, usagesAsTargetQuery);
1115     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
1116     ↪ constants.Break, handler);
1117     foreach (var usage in usages)
1118     {
1119         if (equalityComparer.Equals(links.GetIndex(usage), linkIndex) ||
1120             ↪ !links.Exists(links.GetIndex(usage)))
1121         {
1122             continue;
1123         }
1124         handlerState.Apply(links.Delete(links.GetIndex(usage), handlerState.Handler));
1125     }
1126     return handlerState.Result;
1127 }
1128
1129 /// <summary>
1130 /// <para>
1131 /// Deletes the by query using the specified links.
1132 /// </para>
1133 /// <para></para>
1134 /// </summary>
1135 /// <typeparam name="TLinkAddress">
1136 /// <para>The link.</para>
1137 /// <para></para>
1138 /// </typeparam>
1139 /// <param name="links">
1140 /// <para>The links.</para>
1141 /// <para></para>
1142 /// </param>
1143 /// <param name="query">
1144 /// <para>The query.</para>
1145 /// <para></para>
1146 /// </param>
1147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1148 public static void DeleteByQuery<TLinkAddress>(this ILinks<TLinkAddress> links,
1149     ↪ Link<TLinkAddress> query)

```



```

1141 {
1142     var queryResult = new List<TLinkAddress>();
1143     var queryResultFiller = new ListFiller<TLinkAddress, TLinkAddress>(queryResult,
1144         ↪ links.Constants.Continue);
1145     links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
1146     foreach (var link in queryResult)
1147     {
1148         links.Delete(link);
1149     }
1150 }
1151
1152 // TODO: Move to Platform.Data
1153 /// <summary>
1154 /// <para>
1155 /// Determines whether are values reset.
1156 /// </para>
1157 /// </summary>
1158 /// <typeparam name="TLinkAddress">
1159 /// <para>The link.</para>
1160 /// <para></para>
1161 /// </typeparam>
1162 /// <param name="links">
1163 /// <para>The links.</para>
1164 /// <para></para>
1165 /// </param>
1166 /// <param name="linkIndex">
1167 /// <para>The link index.</para>
1168 /// <para></para>
1169 /// </param>
1170 /// <returns>
1171 /// <para>The bool</para>
1172 /// <para></para>
1173 /// </returns>
1174 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1175 public static bool AreValuesReset<TLinkAddress>(this ILinks<TLinkAddress> links,
1176     ↪ TLinkAddress linkIndex)
1177 {
1178     var nullConstant = links.Constants.Null;
1179     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1180     var link = links.GetLink(linkIndex);
1181     for (int i = 1; i < link.Count; i++)
1182     {
1183         if (!equalityComparer.Equals(link[i], nullConstant))
1184         {
1185             return false;
1186         }
1187     }
1188     return true;
1189 }
1190
1191 public static void ResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1192     ↪ TLinkAddress linkIndex) => links.ResetValues(linkIndex, null);
1193
1194 // TODO: Create a universal version of this method in Platform.Data (with using of for
1195     ↪ loop)
1196 /// <summary>
1197 /// <para>
1198 /// Resets the values using the specified links.
1199 /// </para>
1200 /// <para></para>
1201 /// </summary>
1202 /// <typeparam name="TLinkAddress">
1203 /// <para>The link.</para>
1204 /// <para></para>
1205 /// </typeparam>
1206 /// <param name="links">
1207 /// <para>The links.</para>
1208 /// <para></para>
1209 /// </param>
1210 /// <param name="linkIndex">
1211 /// <para>The link index.</para>
1212 /// <para></para>
1213 /// </param>
1214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1215 public static TLinkAddress ResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1216     ↪ TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1217 {

```

```

1214     var nullConstant = links.Constants.Null;
1215     var updateRequest = new Link<TLinkAddress>(linkIndex, nullConstant, nullConstant);
1216     return links.Update(updateRequest, handler);
1217 }
1218
1219 public static void EnforceResetValues<TLinkAddress>(this ILinks<TLinkAddress> links,
1220     ↪ TLinkAddress linkIndex) => links.EnforceResetValues(linkIndex, null);
1221
1222 // TODO: Create a universal version of this method in Platform.Data (with using of for
1223     ↪ loop)
1224 /// <summary>
1225 /// <para>
1226 /// Enforces the reset values using the specified links.
1227 /// </para>
1228 /// <para></para>
1229 /// </summary>
1230 /// <typeparam name="TLinkAddress">
1231 /// <para>The link.</para>
1232 /// <para></para>
1233 /// </typeparam>
1234 /// <param name="links">
1235 /// <para>The links.</para>
1236 /// <para></para>
1237 /// </param>
1238 /// <param name="linkIndex">
1239 /// <para>The link index.</para>
1240 /// <para></para>
1241 /// </param>
1242 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1243 public static TLinkAddress EnforceResetValues<TLinkAddress>(this ILinks<TLinkAddress>
1244     ↪ links, TLinkAddress linkIndex, WriteHandler<TLinkAddress>? handler)
1245 {
1246     if (!links.AreValuesReset(linkIndex))
1247     {
1248         return links.ResetValues(linkIndex, handler);
1249     }
1250     return links.Constants.Continue;
1251 }
1252
1253 public static void MergeUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
1254     ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex) =>
1255     ↪ links.MergeUsages(oldLinkIndex, newLinkIndex, null);
1256
1257 /// <summary>
1258 /// Merging two usages graphs, all children of old link moved to be children of new link
1259     ↪ or deleted.
1260 /// </summary>
1261 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1262 public static TLinkAddress MergeUsages<TLinkAddress>(this ILinks<TLinkAddress> links,
1263     ↪ TLinkAddress oldLinkIndex, TLinkAddress newLinkIndex, WriteHandler<TLinkAddress>?
1264     ↪ handler)
1265 {
1266     var equalityComparer = EqualityComparer<TLinkAddress>.Default;
1267     if (equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1268     {
1269         return newLinkIndex;
1270     }
1271     var constants = links.Constants;
1272     var usagesAsSource = links.All(new Link<TLinkAddress>(constants.Any, oldLinkIndex,
1273         ↪ constants.Any));
1274     WriteHandlerState<TLinkAddress> handlerState = new(constants.Continue,
1275         ↪ constants.Break, handler);
1276     for (var i = 0; i < usagesAsSource.Count; i++)
1277     {
1278         var usageAsSource = usagesAsSource[i];
1279         if (equalityComparer.Equals(usageAsSource[constants.IndexPart], oldLinkIndex))
1280         {
1281             continue;
1282         }
1283         var restriction = new
1284             ↪ LinkAddress<TLinkAddress>(usageAsSource[constants.IndexPart]);
1285         var substitution = new Link<TLinkAddress>(newLinkIndex,
1286             ↪ usageAsSource[constants.TargetPart]);
1287         handlerState.Apply(links.Update(restriction, substitution,
1288             ↪ handlerState.Handler));
1289     }
1290 }

```

```

1278     var usagesAsTarget = links.All(new Link<TLinkAddress>(constants.Any, constants.Any,
1279         ↪ oldLinkIndex));
1280     for (var i = 0; i < usagesAsTarget.Count; i++)
1281     {
1282         var usageAsTarget = usagesAsTarget[i];
1283         if (equalityComparer.Equals(usageAsTarget[constants.IndexPart], oldLinkIndex))
1284         {
1285             continue;
1286         }
1287         var restriction = links.GetLink(usageAsTarget[constants.IndexPart]);
1288         var substitution = new Link<TLinkAddress>(usageAsTarget[constants.TargetPart],
1289             ↪ newLinkIndex);
1290         handlerState.Apply(links.Update(restriction, substitution,
1291             ↪ 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);

```

```

1344         links = new LinksCascadeUniquenessAndUsagesResolver<TLinkAddress>(links);
1345         return links;
1346     }
1347
1348     /// <summary>
1349     /// <para>
1350     /// Formats the links.
1351     /// </para>
1352     /// <para></para>
1353     /// </summary>
1354     /// <typeparam name="TLinkAddress">
1355     /// <para>The link.</para>
1356     /// <para></para>
1357     /// </typeparam>
1358     /// <param name="links">
1359     /// <para>The links.</para>
1360     /// <para></para>
1361     /// </param>
1362     /// <param name="link">
1363     /// <para>The link.</para>
1364     /// <para></para>
1365     /// </param>
1366     /// <returns>
1367     /// <para>The string</para>
1368     /// <para></para>
1369     /// </returns>
1370     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1371     public static string Format<TLinkAddress>(this ILinks<TLinkAddress> links,
1372     ↪ IList<TLinkAddress>? link)
1373     {
1374         var constants = links.Constants;
1375         return $"({link[constants.IndexPart]}: {link[constants.SourcePart]})
1376         ↪ {link[constants.TargetPart]}";
1377     }
1378
1379     /// <summary>
1380     /// <para>
1381     /// Formats the links.
1382     /// </para>
1383     /// <para></para>
1384     /// </summary>
1385     /// <typeparam name="TLinkAddress">
1386     /// <para>The link.</para>
1387     /// <para></para>
1388     /// </typeparam>
1389     /// <param name="links">
1390     /// <para>The links.</para>
1391     /// <para></para>
1392     /// </param>
1393     /// <param name="link">
1394     /// <para>The link.</para>
1395     /// <para></para>
1396     /// </param>
1397     /// <returns>
1398     /// <para>The string</para>
1399     /// <para></para>
1400     /// </returns>
1401     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1402     public static string Format<TLinkAddress>(this ILinks<TLinkAddress> links, TLinkAddress
1403     ↪ link) => links.Format(links.GetLink(link));
1404 }

```

## 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}"/>

```

```

13     public interface ISynchronizedLinks<TLinkAddress> : ISynchronizedLinks<TLinkAddress,
14         ↳ ILinks<TLinkAddress>, LinksConstants<TLinkAddress>>, ILinks<TLinkAddress>
15     {
16     }

```

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

```

1  using Platform.Collections.Lists;
2  using Platform.Exceptions;
3  using Platform.Ranges;
4  using Platform.Singletons;
5  using System;
6  using System.Collections;
7  using System.Collections.Generic;
8  using System.Runtime.CompilerServices;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets
13 {
14     /// <summary>
15     /// Структура описывающая уникальную связь.
16     /// </summary>
17     public struct Link<TLinkAddress> : IEquatable<Link<TLinkAddress>>,
18         ↳ IReadOnlyList<TLinkAddress>, IList<TLinkAddress>
19     {
20         /// <summary>
21         /// <para>
22         /// The link.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         public static readonly Link<TLinkAddress> Null = new Link<TLinkAddress>();
27         private static readonly LinksConstants<TLinkAddress> _constants =
28             ↳ Default<LinksConstants<TLinkAddress>>.Instance;
29         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
30             ↳ EqualityComparer<TLinkAddress>.Default;
31         private const int Length = 3;
32
33         /// <summary>
34         /// <para>
35         /// The index.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         public readonly TLinkAddress Index;
40         /// <summary>
41         /// <para>
42         /// The source.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         public readonly TLinkAddress Source;
47         /// <summary>
48         /// <para>
49         /// The target.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         public readonly TLinkAddress Target;
54
55         /// <summary>
56         /// <para>
57         /// Initializes a new <see cref="Link"/> instance.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         /// <param name="values">
62         /// <para>A values.</para>
63         /// <para></para>
64         /// </param>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public Link(params TLinkAddress[] values) => SetValues(values, out Index, out Source,
67             ↳ out Target);
68
69         /// <summary>
70         /// <para>
71         /// Initializes a new <see cref="Link"/> instance.
72         /// </para>

```

```

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(ICollection<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 ICollection<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,
    → out Target);
121
122     /// <summary>
123     /// <para>
124     /// Initializes a new <see cref="Link"/> instance.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="index">
129     /// <para>A index.</para>
130     /// <para></para>
131     /// </param>
132     /// <param name="source">
133     /// <para>A source.</para>
134     /// <para></para>
135     /// </param>
136     /// <param name="target">
137     /// <para>A target.</para>
138     /// <para></para>
139     /// </param>
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     public Link(TLinkAddress index, TLinkAddress source, TLinkAddress target)
142     {
143         Index = index;
144         Source = source;

```

```

145     Target = target;
146 }
147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
148 private static void SetValues(ref Link<TLinkAddress> other, out TLinkAddress index, out
    ↪ TLinkAddress source, out TLinkAddress target)
149 {
150     index = other.Index;
151     source = other.Source;
152     target = other.Target;
153 }
154 [MethodImpl(MethodImplOptions.AggressiveInlining)]
155 private static void SetValues(IList<TLinkAddress>? values, out TLinkAddress index, out
    ↪ TLinkAddress source, out TLinkAddress target)
156 {
157     if (values == null)
158     {
159         index = default;
160         source = default;
161         target = default;
162         return;
163     }
164     switch (values.Count)
165     {
166         case 3:
167             index = values[0];
168             source = values[1];
169             target = values[2];
170             break;
171         case 2:
172             index = values[0];
173             source = values[1];
174             target = default;
175             break;
176         case 1:
177             index = values[0];
178             source = default;
179             target = default;
180             break;
181         default:
182             index = default;
183             source = default;
184             target = default;
185             break;
186     }
187 }
188
189 /// <summary>
190 /// <para>
191 /// Gets the hash code.
192 /// </para>
193 /// <para></para>
194 /// </summary>
195 /// <returns>
196 /// <para>The int</para>
197 /// <para></para>
198 /// </returns>
199 [MethodImpl(MethodImplOptions.AggressiveInlining)]
200 public override int GetHashCode() => (Index, Source, Target).GetHashCode();
201
202 /// <summary>
203 /// <para>
204 /// Determines whether this instance is null.
205 /// </para>
206 /// <para></para>
207 /// </summary>
208 /// <returns>
209 /// <para>The bool</para>
210 /// <para></para>
211 /// </returns>
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
    && _equalityComparer.Equals(Source, _constants.Null)
    && _equalityComparer.Equals(Target, _constants.Null);
214
215
216
217 /// <summary>
218 /// <para>
219 /// Determines whether this instance equals.
220 /// </para>
221 /// <para></para>

```

```

222     /// </summary>
223     /// <param name="other">
224     /// <para>The other.</para>
225     /// <para></para>
226     /// </param>
227     /// <returns>
228     /// <para>The bool</para>
229     /// <para></para>
230     /// </returns>
231     [MethodImpl(MethodImplOptions.AggressiveInlining)]
232     public override bool Equals(object other) => other is Link<TLinkAddress> &&
        ↪ Equals((Link<TLinkAddress>)other);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance equals.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="other">
241     /// <para>The other.</para>
242     /// <para></para>
243     /// </param>
244     /// <returns>
245     /// <para>The bool</para>
246     /// <para></para>
247     /// </returns>
248     [MethodImpl(MethodImplOptions.AggressiveInlining)]
249     public bool Equals(Link<TLinkAddress> other) => _equalityComparer.Equals(Index,
        ↪ other.Index)
        && _equalityComparer.Equals(Source, other.Source)
        && _equalityComparer.Equals(Target, other.Target);
250
251
252
253     /// <summary>
254     /// <para>
255     /// Returns the string using the specified index.
256     /// </para>
257     /// <para></para>
258     /// </summary>
259     /// <param name="index">
260     /// <para>The index.</para>
261     /// <para></para>
262     /// </param>
263     /// <param name="source">
264     /// <para>The source.</para>
265     /// <para></para>
266     /// </param>
267     /// <param name="target">
268     /// <para>The target.</para>
269     /// <para></para>
270     /// </param>
271     /// <returns>
272     /// <para>The string</para>
273     /// <para></para>
274     /// </returns>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     public static string ToString(TLinkAddress index, TLinkAddress source, TLinkAddress
        ↪ target) => $"{({index}: {source}->{target})}";
277
278     /// <summary>
279     /// <para>
280     /// Returns the string using the specified source.
281     /// </para>
282     /// <para></para>
283     /// </summary>
284     /// <param name="source">
285     /// <para>The source.</para>
286     /// <para></para>
287     /// </param>
288     /// <param name="target">
289     /// <para>The target.</para>
290     /// <para></para>
291     /// </param>
292     /// <returns>
293     /// <para>The string</para>
294     /// <para></para>
295     /// </returns>

```



```

296 [MethodImpl(MethodImplOptions.AggressiveInlining)]
297 public static string ToString(TLinkAddress source, TLinkAddress target) =>
298     ↳ $"({source}->{target})";
299
300 [MethodImpl(MethodImplOptions.AggressiveInlining)]
301 public static implicit operator TLinkAddress[] (Link<TLinkAddress> link) =>
302     ↳ link.ToArray();
303
304 [MethodImpl(MethodImplOptions.AggressiveInlining)]
305 public static implicit operator Link<TLinkAddress>(TLinkAddress[] linkArray) => new
306     ↳ Link<TLinkAddress>(linkArray);
307
308 /// <summary>
309 /// <para>
310 /// Returns the string.
311 /// </para>
312 /// <para></para>
313 /// </summary>
314 /// <returns>
315 /// <para>The string</para>
316 /// <para></para>
317 /// </returns>
318 [MethodImpl(MethodImplOptions.AggressiveInlining)]
319 public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
320     ↳ ToString(Source, Target) : ToString(Index, Source, Target);
321
322 #region IList
323
324 /// <summary>
325 /// <para>
326 /// Gets the count value.
327 /// </para>
328 /// <para></para>
329 /// </summary>
330 public int Count
331 {
332     [MethodImpl(MethodImplOptions.AggressiveInlining)]
333     get => Length;
334 }
335
336 /// <summary>
337 /// <para>
338 /// Gets the is read only value.
339 /// </para>
340 /// <para></para>
341 /// </summary>
342 public bool IsReadOnly
343 {
344     [MethodImpl(MethodImplOptions.AggressiveInlining)]
345     get => true;
346 }
347
348 /// <summary>
349 /// <para>
350 /// The not supported exception.
351 /// </para>
352 /// <para></para>
353 /// </summary>
354 public TLinkAddress this[int index]
355 {
356     [MethodImpl(MethodImplOptions.AggressiveInlining)]
357     get
358     {
359         Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
360             ↳ nameof(index));
361         if (index == _constants.IndexPart)
362         {
363             return Index;
364         }
365         if (index == _constants.SourcePart)
366         {
367             return Source;
368         }
369         if (index == _constants.TargetPart)
370         {
371             return Target;
372         }
373         throw new NotSupportedException(); // Impossible path due to
374             ↳ 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     /// <summary>
477     /// <para>
478     /// Determines whether this instance remove.
479     /// </para>
480     /// <para></para>
481     /// </summary>
482     /// <param name="item">
483     /// <para>The item.</para>
484     /// <para></para>
485     /// </param>
486     /// <returns>
487     /// <para>The bool</para>
488     /// <para></para>
489     /// </returns>
490     [MethodImpl(MethodImplOptions.AggressiveInlining)]
491     public bool Remove(TLinkAddress item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
492     /// <summary>
493     /// <para>
494     /// Indexes the of using the specified item.
495     /// </para>
496     /// <para></para>
497     /// </summary>
498     /// <param name="item">
499     /// <para>The item.</para>
500     /// <para></para>
501     /// </param>
502     /// <returns>
503     /// <para>The int</para>
504     /// <para></para>
505     /// </returns>
506     [MethodImpl(MethodImplOptions.AggressiveInlining)]
507     public int IndexOf(TLinkAddress item)
508     {
509         if (_equalityComparer.Equals(Index, item))
510         {
511             return _constants.IndexPart;
512         }
513         if (_equalityComparer.Equals(Source, item))
514         {
515             return _constants.SourcePart;
516         }
517         if (_equalityComparer.Equals(Target, item))
518         {
519             return _constants.TargetPart;
520         }
521         return -1;
522     }
523 }

```

```

524
525     /// <summary>
526     /// <para>
527     /// Inserts the index.
528     /// </para>
529     /// <para></para>
530     /// </summary>
531     /// <param name="index">
532     /// <para>The index.</para>
533     /// <para></para>
534     /// </param>
535     /// <param name="item">
536     /// <para>The item.</para>
537     /// <para></para>
538     /// </param>
539     [MethodImpl(MethodImplOptions.AggressiveInlining)]
540     public void Insert(int index, TLinkAddress item) => throw new NotSupportedException();
541
542     /// <summary>
543     /// <para>
544     /// Removes the at using the specified index.
545     /// </para>
546     /// <para></para>
547     /// </summary>
548     /// <param name="index">
549     /// <para>The index.</para>
550     /// <para></para>
551     /// </param>
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]
553     public void RemoveAt(int index) => throw new NotSupportedException();
554
555     [MethodImpl(MethodImplOptions.AggressiveInlining)]
556     public static bool operator ==(Link<TLinkAddress> left, Link<TLinkAddress> right) =>
557         ↪ left.Equals(right);
558
559     [MethodImpl(MethodImplOptions.AggressiveInlining)]
560     public static bool operator !=(Link<TLinkAddress> left, Link<TLinkAddress> right) =>
561         ↪ !(left == right);
562
563     #endregion
564 }
565 }

```

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

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links operator base.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     public abstract class LinksOperatorBase<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         /// </summary>
42     }
43 }

```



```

38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public TLinkAddress ReservedLinks;
42     /// <summary>
43     /// <para>
44     /// The free links.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     public TLinkAddress FreeLinks;
49     /// <summary>
50     /// <para>
51     /// The first free link.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     public TLinkAddress FirstFreeLink;
56     /// <summary>
57     /// <para>
58     /// The root as source.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLinkAddress RootAsSource;
63     /// <summary>
64     /// <para>
65     /// The root as target.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public TLinkAddress RootAsTarget;
70     /// <summary>
71     /// <para>
72     /// The last free link.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public TLinkAddress LastFreeLink;
77     /// <summary>
78     /// <para>
79     /// The reserved.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLinkAddress Reserved8;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equals.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="obj">
92     /// <para>The obj.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override bool Equals(object obj) => obj is LinksHeader<TLinkAddress> linksHeader
101        ↪ ? Equals(linksHeader) : false;
102
103    /// <summary>
104    /// <para>
105    /// Determines whether this instance equals.
106    /// </para>
107    /// <para></para>
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>

```

```

115     /// </returns>
116     [MethodImpl(MethodImplOptions.AggressiveInlining)]
117     public bool Equals(LinksHeader<TLinkAddress> other)
118     => _equalityComparer.Equals(AllocatedLinks, other.AllocatedLinks)
119         && _equalityComparer.Equals(ReservedLinks, other.ReservedLinks)
120         && _equalityComparer.Equals(FreeLinks, other.FreeLinks)
121         && _equalityComparer.Equals(FirstFreeLink, other.FirstFreeLink)
122         && _equalityComparer.Equals(RootAsSource, other.RootAsSource)
123         && _equalityComparer.Equals(RootAsTarget, other.RootAsTarget)
124         && _equalityComparer.Equals>LastFreeLink, other.LastFreeLink)
125         && _equalityComparer.Equals(Reserved8, other.Reserved8);
126
127     /// <summary>
128     /// <para>
129     /// Gets the hash code.
130     /// </para>
131     /// <para></para>
132     /// </summary>
133     /// <returns>
134     /// <para>The int</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public override int GetHashCode() => (AllocatedLinks, ReservedLinks, FreeLinks,
139     ↪ FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();
140
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static bool operator ==(LinksHeader<TLinkAddress> left, LinksHeader<TLinkAddress>
143     ↪ right) => left.Equals(right);
144
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static bool operator !=(LinksHeader<TLinkAddress> left, LinksHeader<TLinkAddress>
147     ↪ right) => !(left == right);
148 }
149 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.Split.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the external links recursionless size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{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;

```

```

40    /// <summary>
41    /// <para>
42    /// The links data parts.
43    /// </para>
44    /// <para></para>
45    /// </summary>
46    protected readonly byte* LinksDataParts;
47    /// <summary>
48    /// <para>
49    /// The links index parts.
50    /// </para>
51    /// <para></para>
52    /// </summary>
53    protected readonly byte* LinksIndexParts;
54    /// <summary>
55    /// <para>
56    /// The header.
57    /// </para>
58    /// <para></para>
59    /// </summary>
60    protected readonly byte* Header;
61
62    /// <summary>
63    /// <para>
64    /// Initializes a new <see
65    ↪ cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
66    /// </para>
67    /// <para></para>
68    /// </summary>
69    /// <param name="constants">
70    /// <para>A constants.</para>
71    /// </param>
72    /// <param name="linksDataParts">
73    /// <para>A links data parts.</para>
74    /// </param>
75    /// <param name="linksIndexParts">
76    /// <para>A links index parts.</para>
77    /// </param>
78    /// <param name="header">
79    /// <para>A header.</para>
80    /// </param>
81    [MethodImpl(MethodImplOptions.AggressiveInlining)]
82    protected
83    ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
84    ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
85    {
86        LinksDataParts = linksDataParts;
87        LinksIndexParts = linksIndexParts;
88        Header = header;
89        Break = constants.Break;
90        Continue = constants.Continue;
91    }
92
93    /// <summary>
94    /// <para>
95    /// Gets the tree root.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <returns>
100    /// <para>The link</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected abstract TLinkAddress GetTreeRoot();
105
106    /// <summary>
107    /// <para>
108    /// Gets the base part value using the specified link.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <param name="link">
113    /// <para>The link.</para>
114    /// </param>

```

```

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
194         ↳ AsRef<LinksHeader<TLinkAddress>>(Header);
195
196     /// <summary>
197     /// <para>
198     /// Gets the link data part reference using the specified link.
199     /// </para>
200     /// <para></para>
201     /// </summary>
202     /// <param name="link">
203     /// <para>The link.</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>
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     /// </param>
225     /// <returns>
226     /// <para>A ref raw link index part of t link</para>
227     /// <para></para>
228     /// </returns>
229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
230     protected virtual ref RawLinkIndexPart<TLinkAddress>
231         ↳ GetLinkIndexPartReference(TLinkAddress link) => ref
232         ↳ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
233         ↳ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
234         ↳ _addressToInt64Converter.Convert(link)));
235
236     /// <summary>
237     /// <para>
238     /// Gets the link values using the specified link index.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="linkIndex">
243     /// <para>The link index.</para>
244     /// </param>
245     /// <returns>
246     /// <para>A list of t link</para>
247     /// <para></para>
248     /// </returns>
249     [MethodImpl(MethodImplOptions.AggressiveInlining)]
250     protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
251     {
252         ref var link = ref GetLinkDataPartReference(linkIndex);
253         return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
254     }
255
256     /// <summary>
257     /// <para>
258     /// Determines whether this instance first is to the left of second.
259     /// </para>
260     /// <para></para>
261     /// </summary>
262     /// <param name="first">
263     /// <para>The first.</para>
264     /// </param>
265     /// </param>

```

```

260    /// <param name="second">
261    /// <para>The second.</para>
262    /// <para></para>
263    /// </param>
264    /// <returns>
265    /// <para>The bool</para>
266    /// <para></para>
267    /// </returns>
268    [MethodImpl(MethodImplOptions.AggressiveInlining)]
269    protected override bool FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
270    {
271        ref var firstLink = ref GetLinkDataPartReference(first);
272        ref var secondLink = ref GetLinkDataPartReference(second);
273        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
274            ↪ secondLink.Source, secondLink.Target);
275    }
276    /// <summary>
277    /// <para>
278    /// Determines whether this instance first is to the right of second.
279    /// </para>
280    /// <para></para>
281    /// </summary>
282    /// <param name="first">
283    /// <para>The first.</para>
284    /// <para></para>
285    /// </param>
286    /// <param name="second">
287    /// <para>The second.</para>
288    /// <para></para>
289    /// </param>
290    /// <returns>
291    /// <para>The bool</para>
292    /// <para></para>
293    /// </returns>
294    [MethodImpl(MethodImplOptions.AggressiveInlining)]
295    protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
296        ↪ second)
297    {
298        ref var firstLink = ref GetLinkDataPartReference(first);
299        ref var secondLink = ref GetLinkDataPartReference(second);
300        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
301            ↪ secondLink.Source, secondLink.Target);
302    }
303    /// <summary>
304    /// <para>
305    /// The zero.
306    /// </para>
307    /// <para></para>
308    /// </summary>
309    public TLinkAddress this[TLinkAddress index]
310    {
311        [MethodImpl(MethodImplOptions.AggressiveInlining)]
312        get
313        {
314            var root = GetTreeRoot();
315            if (GreaterOrEqualThan(index, GetSize(root)))
316            {
317                return Zero;
318            }
319            while (!EqualToZero(root))
320            {
321                var left = GetLeftOrDefault(root);
322                var leftSize = GetSizeOrZero(left);
323                if (LessThan(index, leftSize))
324                {
325                    root = left;
326                    continue;
327                }
328                if (AreEqual(index, leftSize))
329                {
330                    return root;
331                }
332                root = GetRightOrDefault(root);
333                index = Subtract(index, Increment(leftSize));
334            }
335        }
336    }

```

```

334         return Zero; // TODO: Impossible situation exception (only if tree structure
335         ↪ broken)
336     }
337 }
338
339 /// <summary>
340 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
341 ↪ (концом).
342 /// </summary>
343 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
344 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
345 /// <returns>Индекс искомой связи.</returns>
346 [MethodImpl(MethodImplOptions.AggressiveInlining)]
347 public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
348 {
349     var root = GetTreeRoot();
350     while (!EqualToZero(root))
351     {
352         ref var rootLink = ref GetLinkDataPartReference(root);
353         var rootSource = rootLink.Source;
354         var rootTarget = rootLink.Target;
355         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
356             ↪ node.Key < root.Key
357         {
358             root = GetLeftOrDefault(root);
359         }
360         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
361             ↪ node.Key > root.Key
362         {
363             root = GetRightOrDefault(root);
364         }
365         else // node.Key == root.Key
366         {
367             return root;
368         }
369     }
370     return Zero;
371 }
372
373 // TODO: Return indices range instead of references count
374 /// <summary>
375 /// <para>
376 /// Counts the usages using the specified link.
377 /// </para>
378 /// <para></para>
379 /// </summary>
380 /// <param name="link">
381 /// <para>The link.</para>
382 /// <para></para>
383 /// </param>
384 /// <returns>
385 /// <para>The link</para>
386 /// <para></para>
387 /// </returns>
388 [MethodImpl(MethodImplOptions.AggressiveInlining)]
389 public TLinkAddress CountUsages(TLinkAddress link)
390 {
391     var root = GetTreeRoot();
392     var total = GetSize(root);
393     var totalRightIgnore = Zero;
394     while (!EqualToZero(root))
395     {
396         var @base = GetBasePartValue(root);
397         if (LessOrEqualThan(@base, link))
398         {
399             root = GetRightOrDefault(root);
400         }
401         else
402         {
403             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
404             root = GetLeftOrDefault(root);
405         }
406     }
407     root = GetTreeRoot();
408     var totalLeftIgnore = Zero;
409     while (!EqualToZero(root))
410     {
411         var @base = GetBasePartValue(root);

```

```

408         if (GreaterOrEqualThan(@base, link))
409         {
410             root = GetLeftOrDefault(root);
411         }
412         else
413         {
414             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
415             root = GetRightOrDefault(root);
416         }
417     }
418     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
419 }
420
421 /// <summary>
422 /// <para>
423 /// Eaches the usage using the specified base.
424 /// </para>
425 /// <para></para>
426 /// </summary>
427 /// <param name="@base">
428 /// <para>The base.</para>
429 /// <para></para>
430 /// </param>
431 /// <param name="handler">
432 /// <para>The handler.</para>
433 /// <para></para>
434 /// </param>
435 /// <returns>
436 /// <para>The link</para>
437 /// <para></para>
438 /// </returns>
439 [MethodImpl(MethodImplOptions.AggressiveInlining)]
440 public TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
441     ↳ EachUsageCore(@base, GetTreeRoot(), handler);
442
443 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
444 ↳ low-level MSIL stack.
445 [MethodImpl(MethodImplOptions.AggressiveInlining)]
446 private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
447     ↳ ReadHandler<TLinkAddress>? handler)
448 {
449     var @continue = Continue;
450     if (EqualToZero(link))
451     {
452         return @continue;
453     }
454     var linkBasePart = GetBasePartValue(link);
455     var @break = Break;
456     if (GreaterThan(linkBasePart, @base))
457     {
458         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
459         {
460             return @break;
461         }
462     }
463     else if (LessThan(linkBasePart, @base))
464     {
465         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
466         {
467             return @break;
468         }
469     }
470     else //if (linkBasePart == @base)
471     {
472         if (AreEqual(handler(GetLinkValues(link)), @break))
473         {
474             return @break;
475         }
476         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
477         {
478             return @break;
479         }
480         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
481         {
482             return @break;
483         }
484     }
485     return @continue;
486 }

```



```

483     }
484
485     /// <summary>
486     /// <para>
487     /// Prints the node value using the specified node.
488     /// </para>
489     /// <para></para>
490     /// </summary>
491     /// <param name="node">
492     /// <para>The node.</para>
493     /// <para></para>
494     /// </param>
495     /// <param name="sb">
496     /// <para>The sb.</para>
497     /// <para></para>
498     /// </param>
499     [MethodImpl(MethodImplOptions.AggressiveInlining)]
500     protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
501     {
502         ref var link = ref GetLinkDataPartReference(node);
503         sb.Append(' ');
504         sb.Append(link.Source);
505         sb.Append('-');
506         sb.Append('>');
507         sb.Append(link.Target);
508     }
509 }
510 }

```

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

```

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

```

```

48     /// <para>
49     /// The links index parts.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     protected readonly byte* LinksIndexParts;
54     /// <summary>
55     /// <para>
56     /// The header.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     protected readonly byte* Header;
61
62     /// <summary>
63     /// <para>
64     /// Initializes a new <see cref="ExternalLinksSizeBalancedTreeMethodsBase"/> instance.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="constants">
69     /// <para>A constants.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="linksDataParts">
73     /// <para>A links data parts.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="linksIndexParts">
77     /// <para>A links index parts.</para>
78     /// <para></para>
79     /// </param>
80     /// <param name="header">
81     /// <para>A header.</para>
82     /// <para></para>
83     /// </param>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
86     ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
87     {
88         LinksDataParts = linksDataParts;
89         LinksIndexParts = linksIndexParts;
90         Header = header;
91         Break = constants.Break;
92         Continue = constants.Continue;
93     }
94     /// <summary>
95     /// <para>
96     /// Gets the tree root.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <returns>
101    /// <para>The link</para>
102    /// <para></para>
103    /// </returns>
104    [MethodImpl(MethodImplOptions.AggressiveInlining)]
105    protected abstract TLinkAddress GetTreeRoot();
106
107    /// <summary>
108    /// <para>
109    /// Gets the base part value using the specified link.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="link">
114    /// <para>The link.</para>
115    /// <para></para>
116    /// </param>
117    /// <returns>
118    /// <para>The link</para>
119    /// <para></para>
120    /// </returns>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    protected abstract TLinkAddress GetBasePartValue(TLinkAddress link);
123
124    /// <summary>

```

```

125    /// <para>
126    /// Determines whether this instance first is to the right of second.
127    /// </para>
128    /// <para></para>
129    /// </summary>
130    /// <param name="source">
131    /// <para>The source.</para>
132    /// <para></para>
133    /// </param>
134    /// <param name="target">
135    /// <para>The target.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="rootSource">
139    /// <para>The root source.</para>
140    /// <para></para>
141    /// </param>
142    /// <param name="rootTarget">
143    /// <para>The root target.</para>
144    /// <para></para>
145    /// </param>
146    /// <returns>
147    /// <para>The bool</para>
148    /// <para></para>
149    /// </returns>
150    [MethodImpl(MethodImplOptions.AggressiveInlining)]
151    protected abstract bool FirstIsToTheRightOfSecond(TLinkAddress source, TLinkAddress
    ↪ target, TLinkAddress rootSource, TLinkAddress rootTarget);

152    /// <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     /// </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>
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)

```

```

270 {
271     ref var firstLink = ref GetLinkDataPartReference(first);
272     ref var secondLink = ref GetLinkDataPartReference(second);
273     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
274 }
275
276 /// <summary>
277 /// <para>
278 /// Determines whether this instance first is to the right of second.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 /// <param name="first">
283 /// <para>The first.</para>
284 /// <para></para>
285 /// </param>
286 /// <param name="second">
287 /// <para>The second.</para>
288 /// <para></para>
289 /// </param>
290 /// <returns>
291 /// <para>The bool</para>
292 /// <para></para>
293 /// </returns>
294 [MethodImpl(MethodImplOptions.AggressiveInlining)]
295 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
    ↪ second)
296 {
297     ref var firstLink = ref GetLinkDataPartReference(first);
298     ref var secondLink = ref GetLinkDataPartReference(second);
299     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
300 }
301
302 /// <summary>
303 /// <para>
304 /// The zero.
305 /// </para>
306 /// <para></para>
307 /// </summary>
308 public TLinkAddress this[TLinkAddress index]
309 {
310     [MethodImpl(MethodImplOptions.AggressiveInlining)]
311     get
312     {
313         var root = GetTreeRoot();
314         if (GreaterOrEqualThan(index, GetSize(root)))
315         {
316             return Zero;
317         }
318         while (!EqualToZero(root))
319         {
320             var left = GetLeftOrDefault(root);
321             var leftSize = GetSizeOrZero(left);
322             if (LessThan(index, leftSize))
323             {
324                 root = left;
325                 continue;
326             }
327             if (AreEqual(index, leftSize))
328             {
329                 return root;
330             }
331             root = GetRightOrDefault(root);
332             index = Subtract(index, Increment(leftSize));
333         }
334         return Zero; // TODO: Impossible situation exception (only if tree structure
            ↪ broken)
335     }
336 }
337
338 /// <summary>
339 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
340 /// </summary>
341 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
342 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>

```

```

343 /// <returns>Индекс искомой связи.</returns>
344 [MethodImpl(MethodImplOptions.AggressiveInlining)]
345 public TLinkAddress Search(TLinkAddress source, TLinkAddress target)
346 {
347     var root = GetTreeRoot();
348     while (!EqualToZero(root))
349     {
350         ref var rootLink = ref GetLinkDataPartReference(root);
351         var rootSource = rootLink.Source;
352         var rootTarget = rootLink.Target;
353         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
354             ↪ node.Key < root.Key
355         {
356             root = GetLeftOrDefault(root);
357         }
358         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
359             ↪ node.Key > root.Key
360         {
361             root = GetRightOrDefault(root);
362         }
363         else // node.Key == root.Key
364         {
365             return root;
366         }
367     }
368     return Zero;
369 }
370
371 /// TODO: Return indices range instead of references count
372 /// <summary>
373 /// <para>
374 /// Counts the usages using the specified link.
375 /// </para>
376 /// <para></para>
377 /// </summary>
378 /// <param name="link">
379 /// <para>The link.</para>
380 /// <para></para>
381 /// </param>
382 /// <returns>
383 /// <para>The link</para>
384 /// <para></para>
385 /// </returns>
386 [MethodImpl(MethodImplOptions.AggressiveInlining)]
387 public TLinkAddress CountUsages(TLinkAddress link)
388 {
389     var root = GetTreeRoot();
390     var total = GetSize(root);
391     var totalRightIgnore = Zero;
392     while (!EqualToZero(root))
393     {
394         var @base = GetBasePartValue(root);
395         if (LessOrEqualThan(@base, link))
396         {
397             root = GetRightOrDefault(root);
398         }
399         else
400         {
401             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
402             root = GetLeftOrDefault(root);
403         }
404     }
405     root = GetTreeRoot();
406     var totalLeftIgnore = Zero;
407     while (!EqualToZero(root))
408     {
409         var @base = GetBasePartValue(root);
410         if (GreaterOrEqualThan(@base, link))
411         {
412             root = GetLeftOrDefault(root);
413         }
414         else
415         {
416             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
417             root = GetRightOrDefault(root);
418         }
419     }
420     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);

```

```

419     }
420
421     /// <summary>
422     /// <para>
423     /// Eaches the usage using the specified base.
424     /// </para>
425     /// <para></para>
426     /// </summary>
427     /// <param name="@base">
428     /// <para>The base.</para>
429     /// <para></para>
430     /// </param>
431     /// <param name="handler">
432     /// <para>The handler.</para>
433     /// <para></para>
434     /// </param>
435     /// <returns>
436     /// <para>The link</para>
437     /// <para></para>
438     /// </returns>
439     [MethodImpl(MethodImplOptions.AggressiveInlining)]
440     public TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
441         ↪ EachUsageCore(@base, GetTreeRoot(), handler);
442
443     // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
444     ↪ low-level MSIL stack.
445     [MethodImpl(MethodImplOptions.AggressiveInlining)]
446     private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
447         ↪ ReadHandler<TLinkAddress>? handler)
448     {
449         var @continue = Continue;
450         if (EqualToZero(link))
451         {
452             return @continue;
453         }
454         var linkBasePart = GetBasePartValue(link);
455         var @break = Break;
456         if (GreaterThan(linkBasePart, @base))
457         {
458             if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
459             {
460                 return @break;
461             }
462         }
463         else if (LessThan(linkBasePart, @base))
464         {
465             if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
466             {
467                 return @break;
468             }
469         }
470         else //if (linkBasePart == @base)
471         {
472             if (AreEqual(handler(GetLinkValues(link)), @break))
473             {
474                 return @break;
475             }
476             if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
477             {
478                 return @break;
479             }
480             if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
481             {
482                 return @break;
483             }
484         }
485         return @continue;
486     }
487
488     /// <summary>
489     /// <para>
490     /// Prints the node value using the specified node.
491     /// </para>
492     /// <para></para>
493     /// </summary>
494     /// <param name="node">
495     /// <para>The node.</para>
496     /// <para></para>
497     /// </param>

```

```

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)]

```



```

56     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
57         ↪ GetLinkIndexPartReference(node).LeftAsSource;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
75         ↪ GetLinkIndexPartReference(node).RightAsSource;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected override TLinkAddress GetLeft(TLinkAddress node) =>
93         ↪ GetLinkIndexPartReference(node).LeftAsSource;
94
95     /// <summary>
96     /// <para>
97     /// Gets the right using the specified node.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="node">
102    /// <para>The node.</para>
103    /// <para></para>
104    /// </param>
105    /// <returns>
106    /// <para>The link</para>
107    /// <para></para>
108    /// </returns>
109    [MethodImpl(MethodImplOptions.AggressiveInlining)]
110    protected override TLinkAddress GetRight(TLinkAddress node) =>
111        ↪ GetLinkIndexPartReference(node).RightAsSource;
112
113    /// <summary>
114    /// <para>
115    /// Sets the left using the specified node.
116    /// </para>
117    /// <para></para>
118    /// </summary>
119    /// <param name="node">
120    /// <para>The node.</para>
121    /// <para></para>
122    /// </param>
123    /// <param name="left">
124    /// <para>The left.</para>
125    /// <para></para>
126    /// </param>
127    [MethodImpl(MethodImplOptions.AggressiveInlining)]
128    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
129        ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
130
131    /// <summary>
132    /// <para>

```

```

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

```

```

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

```

```

273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLinkAddress node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);
279         link.LeftAsSource = Zero;
280         link.RightAsSource = Zero;
281         link.SizeAsSource = Zero;
282     }
283 }
284 }

```

### 1.35 ./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 }

```

```

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).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>

```

```

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

```

```

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

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

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

```

```

280         link.RightAsSource = Zero;
281         link.SizeAsSource = Zero;
282     }
283 }
284 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.Split.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the external links targets recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
14     public unsafe class ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLinkAddress> :
15         ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<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).LeftAsTarget;
62
63         /// <summary>
64         /// <para>
65         /// Gets the right reference using the specified node.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         /// <param name="node">
70         /// <para>The node.</para>
71         /// <para></para>
72         /// </param>
73         /// <returns>
74         /// <para>The ref link</para>
75         /// <para></para>
76         /// </returns>
77         [MethodImpl(MethodImplOptions.AggressiveInlining)]
78         protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
79         ↪ GetLinkIndexPartReference(node).RightAsTarget;
80     }
81 }

```



```

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

```

```

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

```

```

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

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

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

```

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

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

```

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

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

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

```

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

```

1 using System;
2 using System.Text;
3 using System.Collections.Generic;
4 using System.Runtime.CompilerServices;
5 using Platform.Collections.Methods.Trees;

```

```

6 using Platform.Converters;
7 using Platform.Delegates;
8 using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.Split.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the internal links recursionless size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLinkAddress}"/>
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class
23     ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<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         /// <para></para>
66         /// </summary>
67         protected readonly byte* Header;
68
69         /// <summary>
70         /// <para>
71         /// Initializes a new <see
72         ↪ cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
73         /// </para>
74         /// <para></para>
75         /// </summary>
76         /// <param name="constants">
77         /// <para>A constants.</para>
78         /// <para></para>
79         /// </param>
80         /// <param name="linksDataParts">
81         /// <para>A links data parts.</para>
82         /// <para></para>
83         /// </param>
84         /// <param name="linksIndexParts">
85         /// <para>A links index parts.</para>
86         /// <para></para>
87         /// </param>
88         /// <param name="header">

```



```

81     /// <para>A header.</para>
82     /// <para></para>
83     /// </param>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected
86     ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
87     ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
88     {
89         LinksDataParts = linksDataParts;
90         LinksIndexParts = linksIndexParts;
91         Header = header;
92         Break = constants.Break;
93         Continue = constants.Continue;
94     }
95
96     /// <summary>
97     /// <para>
98     /// Gets the tree root using the specified link.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    /// <param name="link">
103    /// <para>The link.</para>
104    /// <para></para>
105    /// </param>
106    /// <returns>
107    /// <para>The link</para>
108    /// <para></para>
109    /// </returns>
110    [MethodImpl(MethodImplOptions.AggressiveInlining)]
111    protected abstract TLinkAddress GetTreeRoot(TLinkAddress link);
112
113    /// <summary>
114    /// <para>
115    /// Gets the base part value using the specified link.
116    /// </para>
117    /// <para></para>
118    /// </summary>
119    /// <param name="link">
120    /// <para>The link.</para>
121    /// <para></para>
122    /// </param>
123    /// <returns>
124    /// <para>The link</para>
125    /// <para></para>
126    /// </returns>
127    [MethodImpl(MethodImplOptions.AggressiveInlining)]
128    protected abstract TLinkAddress GetBasePartValue(TLinkAddress link);
129
130    /// <summary>
131    /// <para>
132    /// Gets the key part value using the specified link.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="link">
137    /// <para>The link.</para>
138    /// <para></para>
139    /// </param>
140    /// <returns>
141    /// <para>The link</para>
142    /// <para></para>
143    /// </returns>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected abstract TLinkAddress GetKeyPartValue(TLinkAddress link);
146
147    /// <summary>
148    /// <para>
149    /// Gets the link data part reference using the specified link.
150    /// </para>
151    /// <para></para>
152    /// </summary>
153    /// <param name="link">
154    /// <para>The link.</para>
155    /// <para></para>
156    /// </param>
157    /// <returns>
158    /// <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>
    ↳ GetLinkDataPartReference(TLinkAddress link) => ref
    ↳ AsRef<RawLinkDataPart<TLinkAddress>>(LinksDataParts +
    ↳ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));
161
162 /// <summary>
163 /// <para>
164 /// Gets the link index part reference using the specified link.
165 /// </para>
166 /// <para></para>
167 /// </summary>
168 /// <param name="link">
169 /// <para>The link.</para>
170 /// <para></para>
171 /// </param>
172 /// </returns>
173 /// <para>A ref raw link index part of t link</para>
174 /// <para></para>
175 /// </returns>
176 [MethodImpl(MethodImplOptions.AggressiveInlining)]
177 protected virtual ref RawLinkIndexPart<TLinkAddress>
    ↳ GetLinkIndexPartReference(TLinkAddress link) => ref
    ↳ AsRef<RawLinkIndexPart<TLinkAddress>>(LinksIndexParts +
    ↳ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));
178
179 /// <summary>
180 /// <para>
181 /// Determines whether this instance first is to the left of second.
182 /// </para>
183 /// <para></para>
184 /// </summary>
185 /// <param name="first">
186 /// <para>The first.</para>
187 /// <para></para>
188 /// </param>
189 /// <param name="second">
190 /// <para>The second.</para>
191 /// <para></para>
192 /// </param>
193 /// </returns>
194 /// <para>The bool</para>
195 /// <para></para>
196 /// </returns>
197 [MethodImpl(MethodImplOptions.AggressiveInlining)]
198 protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress
    ↳ second) => LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
199
200 /// <summary>
201 /// <para>
202 /// Determines whether this instance first is to the right of second.
203 /// </para>
204 /// <para></para>
205 /// </summary>
206 /// <param name="first">
207 /// <para>The first.</para>
208 /// <para></para>
209 /// </param>
210 /// <param name="second">
211 /// <para>The second.</para>
212 /// <para></para>
213 /// </param>
214 /// </returns>
215 /// <para>The bool</para>
216 /// <para></para>
217 /// </returns>
218 [MethodImpl(MethodImplOptions.AggressiveInlining)]
219 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
    ↳ second) => GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
220
221 /// <summary>
222 /// <para>
223 /// Gets the link values using the specified link index.
224 /// </para>

```

```

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

```

```

301     /// <returns>
302     /// <para>The zero.</para>
303     /// <para></para>
304     /// </returns>
305     [MethodImpl(MethodImplOptions.AggressiveInlining)]
306     protected TLinkAddress SearchCore(TLinkAddress root, TLinkAddress key)
307     {
308         while (!EqualToZero(root))
309         {
310             var rootKey = GetKeyPartValue(root);
311             if (LessThan(key, rootKey)) // node.Key < root.Key
312             {
313                 root = GetLeftOrDefault(root);
314             }
315             else if (GreaterThan(key, rootKey)) // node.Key > root.Key
316             {
317                 root = GetRightOrDefault(root);
318             }
319             else // node.Key == root.Key
320             {
321                 return root;
322             }
323         }
324         return Zero;
325     }
326
327     // TODO: Return indices range instead of references count
328     /// <summary>
329     /// <para>
330     /// Counts the usages using the specified link.
331     /// </para>
332     /// <para></para>
333     /// </summary>
334     /// <param name="link">
335     /// <para>The link.</para>
336     /// <para></para>
337     /// </param>
338     /// <returns>
339     /// <para>The link</para>
340     /// <para></para>
341     /// </returns>
342     [MethodImpl(MethodImplOptions.AggressiveInlining)]
343     public TLinkAddress CountUsages(TLinkAddress link) => GetSizeOrZero(GetTreeRoot(link));
344
345     /// <summary>
346     /// <para>
347     /// Eaches the usage using the specified base.
348     /// </para>
349     /// <para></para>
350     /// </summary>
351     /// <param name="@base">
352     /// <para>The base.</para>
353     /// <para></para>
354     /// </param>
355     /// <param name="handler">
356     /// <para>The handler.</para>
357     /// <para></para>
358     /// </param>
359     /// <returns>
360     /// <para>The link</para>
361     /// <para></para>
362     /// </returns>
363     [MethodImpl(MethodImplOptions.AggressiveInlining)]
364     public TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
365         ↳ EachUsageCore(@base, GetTreeRoot(@base), handler);
366
367     // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
368     ↳ low-level MSIL stack.
369     [MethodImpl(MethodImplOptions.AggressiveInlining)]
370     private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
371         ↳ ReadHandler<TLinkAddress>? handler)
372     {
373         var @continue = Continue;
374         if (EqualToZero(link))
375         {
376             return @continue;
377         }
378         var @break = Break;

```

```

376         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
377         {
378             return @break;
379         }
380         if (AreEqual(handler(GetLinkValues(link)), @break))
381         {
382             return @break;
383         }
384         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
385         {
386             return @break;
387         }
388         return @continue;
389     }
390
391     /// <summary>
392     /// <para>
393     /// Prints the node value using the specified node.
394     /// </para>
395     /// <para></para>
396     /// </summary>
397     /// <param name="node">
398     /// <para>The node.</para>
399     /// <para></para>
400     /// </param>
401     /// <param name="sb">
402     /// <para>The sb.</para>
403     /// <para></para>
404     /// </param>
405     [MethodImpl(MethodImplOptions.AggressiveInlining)]
406     protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
407     {
408         ref var link = ref GetLinkDataPartReference(node);
409         sb.Append(' ');
410         sb.Append(link.Source);
411         sb.Append('-');
412         sb.Append('>');
413         sb.Append(link.Target);
414     }
415 }
416 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.Split.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the internal links size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizeBalancedTreeMethods{TLinkAddress}"/>
21     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
22     public unsafe abstract class InternalLinksSizeBalancedTreeMethodsBase<TLinkAddress> :
23     ↪ SizeBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
24     {
25         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
26         ↪ = UncheckedConverter<TLinkAddress, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLinkAddress Break;
35     }
36 }

```

```

35     /// The continue.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected readonly TLinkAddress Continue;
40     /// <summary>
41     /// <para>
42     /// The links data parts.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     protected readonly byte* LinksDataParts;
47     /// <summary>
48     /// <para>
49     /// The links index parts.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     protected readonly byte* LinksIndexParts;
54     /// <summary>
55     /// <para>
56     /// The header.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     protected readonly byte* Header;
61
62     /// <summary>
63     /// <para>
64     /// Initializes a new <see cref="InternalLinksSizeBalancedTreeMethodsBase"/> instance.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="constants">
69     /// <para>A constants.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="linksDataParts">
73     /// <para>A links data parts.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="linksIndexParts">
77     /// <para>A links index parts.</para>
78     /// <para></para>
79     /// </param>
80     /// <param name="header">
81     /// <para>A header.</para>
82     /// <para></para>
83     /// </param>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
86         ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
87     {
88         LinksDataParts = linksDataParts;
89         LinksIndexParts = linksIndexParts;
90         Header = header;
91         Break = constants.Break;
92         Continue = constants.Continue;
93     }
94     /// <summary>
95     /// <para>
96     /// Gets the tree root using the specified link.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="link">
101    /// <para>The link.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected abstract TLinkAddress GetTreeRoot(TLinkAddress link);
110
111    /// <summary>

```

```

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

```

```

182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="first">
186     /// <para>The first.</para>
187     /// <para></para>
188     /// </param>
189     /// <param name="second">
190     /// <para>The second.</para>
191     /// <para></para>
192     /// </param>
193     /// <returns>
194     /// <para>The bool</para>
195     /// <para></para>
196     /// </returns>
197     [MethodImpl(MethodImplOptions.AggressiveInlining)]
198     protected override bool FirstIsToTheLeftOfSecond(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         }
259     }

```



```

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

```

```

334    /// <param name="link">
335    /// <para>The link.</para>
336    /// <para></para>
337    /// </param>
338    /// <returns>
339    /// <para>The link</para>
340    /// <para></para>
341    /// </returns>
342    [MethodImpl(MethodImplOptions.AggressiveInlining)]
343    public TLinkAddress CountUsages(TLinkAddress link) => GetSizeOrZero(GetTreeRoot(link));
344
345    /// <summary>
346    /// <para>
347    /// Eaches the usage using the specified base.
348    /// </para>
349    /// <para></para>
350    /// </summary>
351    /// <param name="@base">
352    /// <para>The base.</para>
353    /// <para></para>
354    /// </param>
355    /// <param name="handler">
356    /// <para>The handler.</para>
357    /// <para></para>
358    /// </param>
359    /// <returns>
360    /// <para>The link</para>
361    /// <para></para>
362    /// </returns>
363    [MethodImpl(MethodImplOptions.AggressiveInlining)]
364    public TLinkAddress EachUsage(TLinkAddress @base, ReadHandler<TLinkAddress>? handler) =>
365        ↳ EachUsageCore(@base, GetTreeRoot(@base), handler);
366
367    // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
368    ↳ low-level MSIL stack.
369    [MethodImpl(MethodImplOptions.AggressiveInlining)]
370    private TLinkAddress EachUsageCore(TLinkAddress @base, TLinkAddress link,
371        ↳ ReadHandler<TLinkAddress>? handler)
372    {
373        var @continue = Continue;
374        if (EqualToZero(link))
375        {
376            return @continue;
377        }
378        var @break = Break;
379        if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
380        {
381            return @break;
382        }
383        if (AreEqual(handler(GetLinkValues(link)), @break))
384        {
385            return @break;
386        }
387        if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
388        {
389            return @break;
390        }
391        return @continue;
392    }
393
394    /// <summary>
395    /// <para>
396    /// Prints the node value using the specified node.
397    /// </para>
398    /// <para></para>
399    /// </summary>
400    /// <param name="node">
401    /// <para>The node.</para>
402    /// <para></para>
403    /// </param>
404    /// <param name="sb">
405    /// <para>The sb.</para>
406    /// <para></para>
407    /// </param>
408    [MethodImpl(MethodImplOptions.AggressiveInlining)]
409    protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
410    {
411        ref var link = ref GetLinkDataPartReference(node);

```

```

409         sb.Append(' ');
410         sb.Append(link.Source);
411         sb.Append('-');
412         sb.Append('>');
413         sb.Append(link.Target);
414     }
415 }
416 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections.Methods.Lists;
5  using Platform.Converters;
6  using Platform.Delegates;
7  using static System.Runtime.CompilerServices.Unsafe;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Memory.Split.Generic
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the internal links sources linked list methods.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="RelativeCircularDoublyLinkedListMethods{TLinkAddress}"/>
20     public unsafe class InternalLinksSourcesLinkedListMethods<TLinkAddress> :
21         ↪ RelativeCircularDoublyLinkedListMethods<TLinkAddress>
22     {
23         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
24             ↪ = UncheckedConverter<TLinkAddress, long>.Default;
25         private readonly byte* _linksDataParts;
26         private readonly byte* _linksIndexParts;
27         /// <summary>
28         /// <para>
29         /// The break.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected readonly TLinkAddress Break;
34         /// <summary>
35         /// <para>
36         /// The continue.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         protected readonly TLinkAddress Continue;
41
42         /// <summary>
43         /// <para>
44         /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> 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         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         public InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress> constants,
62             ↪ byte* linksDataParts, byte* linksIndexParts)
63         {
64             _linksDataParts = linksDataParts;
65             _linksIndexParts = linksIndexParts;
66             Break = constants.Break;
67             Continue = constants.Continue;
68         }
69     }
70 }

```

```

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>
83     ↪ GetLinkDataPartReference(TLinkAddress link) => ref
84     ↪ AsRef<RawLinkDataPart<TLinkAddress>>(_linksDataParts +
85     ↪ (RawLinkDataPart<TLinkAddress>.SizeInBytes *
86     ↪ _addressToInt64Converter.Convert(link)));
87
88     /// <summary>
89     /// <para>
90     /// Gets the link index part reference using the specified link.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="link">
95     /// <para>The link.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>A ref raw link index part of t link</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected virtual ref RawLinkIndexPart<TLinkAddress>
104    ↪ GetLinkIndexPartReference(TLinkAddress link) => ref
105    ↪ AsRef<RawLinkIndexPart<TLinkAddress>>(_linksIndexParts +
106    ↪ (RawLinkIndexPart<TLinkAddress>.SizeInBytes *
107    ↪ _addressToInt64Converter.Convert(link)));
108
109    /// <summary>
110    /// <para>
111    /// Gets the first using the specified head.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="head">
116    /// <para>The head.</para>
117    /// <para></para>
118    /// </param>
119    /// <returns>
120    /// <para>The link</para>
121    /// <para></para>
122    /// </returns>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override TLinkAddress GetFirst(TLinkAddress head) =>
125    ↪ GetLinkIndexPartReference(head).RootAsSource;
126
127    /// <summary>
128    /// <para>
129    /// Gets the last using the specified head.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="head">
134    /// <para>The head.</para>
135    /// <para></para>
136    /// </param>
137    /// <returns>
138    /// <para>The link</para>
139    /// <para></para>
140    /// </returns>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected override TLinkAddress GetLast(TLinkAddress head)
143    {
144        var first = GetLinkIndexPartReference(head).RootAsSource;

```

```

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

```

```

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

```

```

286 /// <param name="size">
287 /// <para>The size.</para>
288 /// <para></para>
289 /// </param>
290 [MethodImpl(MethodImplOptions.AggressiveInlining)]
291 protected override void SetSize(TLinkAddress head, TLinkAddress size) =>
    ↳ GetLinkIndexPartReference(head).SizeAsSource = size;

292
293 /// <summary>
294 /// <para>
295 /// Counts the usages using the specified head.
296 /// </para>
297 /// <para></para>
298 /// </summary>
299 /// <param name="head">
300 /// <para>The head.</para>
301 /// <para></para>
302 /// </param>
303 /// <returns>
304 /// <para>The link</para>
305 /// <para></para>
306 /// </returns>
307 [MethodImpl(MethodImplOptions.AggressiveInlining)]
308 public TLinkAddress CountUsages(TLinkAddress head) => GetSize(head);
309
310 /// <summary>
311 /// <para>
312 /// Gets the link values using the specified link index.
313 /// </para>
314 /// <para></para>
315 /// </summary>
316 /// <param name="linkIndex">
317 /// <para>The link index.</para>
318 /// <para></para>
319 /// </param>
320 /// <returns>
321 /// <para>A list of t link</para>
322 /// <para></para>
323 /// </returns>
324 [MethodImpl(MethodImplOptions.AggressiveInlining)]
325 protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
326 {
327     ref var link = ref GetLinkDataPartReference(linkIndex);
328     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
329 }
330
331 /// <summary>
332 /// <para>
333 /// Eaches the usage using the specified source.
334 /// </para>
335 /// <para></para>
336 /// </summary>
337 /// <param name="source">
338 /// <para>The source.</para>
339 /// <para></para>
340 /// </param>
341 /// <param name="handler">
342 /// <para>The handler.</para>
343 /// <para></para>
344 /// </param>
345 /// <returns>
346 /// <para>The continue.</para>
347 /// <para></para>
348 /// </returns>
349 [MethodImpl(MethodImplOptions.AggressiveInlining)]
350 public TLinkAddress EachUsage(TLinkAddress source, ReadHandler<TLinkAddress>? handler)
351 {
352     var @continue = Continue;
353     var @break = Break;
354     var current = GetFirst(source);
355     var first = current;
356     while (!EqualToZero(current))
357     {
358         if (AreEqual(handler(GetLinkValues(current)), @break))
359         {
360             return @break;
361         }
362         current = GetNext(current);

```

```

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

```

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

```

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

```



```

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

```

```

137     /// <para>The right.</para>
138     /// <para></para>
139     /// </param>
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
142         ↪ GetLinkIndexPartReference(node).RightAsSource = right;
143
144     /// <summary>
145     /// <para>
146     /// Gets the size using the specified node.
147     /// </para>
148     /// </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     /// </summary>
166     /// <param name="node">
167     /// <para>The node.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="size">
171     /// <para>The size.</para>
172     /// <para></para>
173     /// </param>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
176         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
177
178     /// <summary>
179     /// <para>
180     /// Gets the tree root using the specified link.
181     /// </para>
182     /// </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     /// </summary>
200     /// <param name="link">
201     /// <para>The link.</para>
202     /// <para></para>
203     /// </param>
204     /// <returns>
205     /// <para>The link</para>
206     /// <para></para>
207     /// </returns>
208     [MethodImpl(MethodImplOptions.AggressiveInlining)]
209     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
210         ↪ GetLinkDataPartReference(link).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) =>
227         ↪ GetLinkDataPartReference(link).Target;
228
229     /// <summary>
230     /// <para>
231     /// Clears the node using the specified node.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240     protected override void ClearNode(TLinkAddress node)
241     {
242         ref var link = ref GetLinkIndexPartReference(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);

```

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

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

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

74
75    /// <summary>
76    /// <para>
77    /// Gets the left using the specified node.
78    /// </para>
79    /// <para></para>
80    /// </summary>
81    /// <param name="node">
82    /// <para>The node.</para>
83    /// <para></para>
84    /// </param>
85    /// <returns>
86    /// <para>The link</para>
87    /// <para></para>
88    /// </returns>

```

```

89 [MethodImpl(MethodImplOptions.AggressiveInlining)]
90 protected override TLinkAddress GetLeft(TLinkAddress node) =>
    ↳ GetLinkIndexPartReference(node).LeftAsSource;
91
92 /// <summary>
93 /// <para>
94 /// Gets the right using the specified node.
95 /// </para>
96 /// <para></para>
97 /// </summary>
98 /// <param name="node">
99 /// <para>The node.</para>
100 /// <para></para>
101 /// </param>
102 /// <returns>
103 /// <para>The link</para>
104 /// <para></para>
105 /// </returns>
106 [MethodImpl(MethodImplOptions.AggressiveInlining)]
107 protected override TLinkAddress GetRight(TLinkAddress node) =>
    ↳ GetLinkIndexPartReference(node).RightAsSource;
108
109 /// <summary>
110 /// <para>
111 /// Sets the left using the specified node.
112 /// </para>
113 /// <para></para>
114 /// </summary>
115 /// <param name="node">
116 /// <para>The node.</para>
117 /// <para></para>
118 /// </param>
119 /// <param name="left">
120 /// <para>The left.</para>
121 /// <para></para>
122 /// </param>
123 [MethodImpl(MethodImplOptions.AggressiveInlining)]
124 protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
    ↳ GetLinkIndexPartReference(node).LeftAsSource = left;
125
126 /// <summary>
127 /// <para>
128 /// Sets the right using the specified node.
129 /// </para>
130 /// <para></para>
131 /// </summary>
132 /// <param name="node">
133 /// <para>The node.</para>
134 /// <para></para>
135 /// </param>
136 /// <param name="right">
137 /// <para>The right.</para>
138 /// <para></para>
139 /// </param>
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
    ↳ GetLinkIndexPartReference(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) =>
    ↳ GetLinkIndexPartReference(node).SizeAsSource;
159
160 /// <summary>

```

```

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

```

```

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

```

#### 1.43 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/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)]

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```



```

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

```

```

216     /// </returns>
217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
218     protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
219         ↳ AsRef<LinksHeader<TLinkAddress>>(_header);
220
221     /// <summary>
222     /// <para>
223     /// Gets the link data part reference using the specified link index.
224     /// </para>
225     /// <para></para>
226     /// </summary>
227     /// <param name="linkIndex">
228     /// <para>The link index.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>A ref raw link data part of t link</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override ref RawLinkDataPart<TLinkAddress>
237         ↳ GetLinkDataPartReference(TLinkAddress linkIndex) => ref
238         ↳ AsRef<RawLinkDataPart<TLinkAddress>>(_linksDataParts + (LinkDataPartSizeInBytes *
239         ↳ ConvertToInt64(linkIndex)));
240
241     /// <summary>
242     /// <para>
243     /// Gets the link index part reference using the specified link index.
244     /// </para>
245     /// <para></para>
246     /// </summary>
247     /// <param name="linkIndex">
248     /// <para>The link index.</para>
249     /// <para></para>
250     /// </param>
251     /// <returns>
252     /// <para>A ref raw link index part of t link</para>
253     /// <para></para>
254     /// </returns>
255     [MethodImpl(MethodImplOptions.AggressiveInlining)]
256     protected override ref RawLinkIndexPart<TLinkAddress>
257         ↳ GetLinkIndexPartReference(TLinkAddress linkIndex) => ref
258         ↳ AsRef<RawLinkIndexPart<TLinkAddress>>(_linksIndexParts + (LinkIndexPartSizeInBytes *
259         ↳ ConvertToInt64(linkIndex)));
260 }
261 }

```

#### 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,
25         ↳ ILinks<TLinkAddress> where TLinkAddress : struct
26     {
27         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
28             ↳ EqualityComparer<TLinkAddress>.Default;
29         private static readonly Comparer<TLinkAddress> _comparer =
30             ↳ Comparer<TLinkAddress>.Default;
31         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
32             ↳ = UncheckedConverter<TLinkAddress, long>.Default;

```

```

private static readonly UncheckedConverter<long, TLinkAddress> _int64ToAddressConverter
    ↪ = UncheckedConverter<long, TLinkAddress>.Default;
private static readonly TLinkAddress _zero = default;
private static readonly TLinkAddress _one = Arithmetic.Increment(_zero);

/// <summary>Возвращает размер одной связи в байтах.</summary>
/// <remarks>
/// Используется только во вне класса, не рекомендуется использовать внутри.
/// Так как во вне не обязательно будет доступен unsafe C#.
/// </remarks>
public static readonly long LinkDataPartSizeInBytes =
    ↪ RawLinkDataPart<TLinkAddress>.SizeInBytes;

/// <summary>
/// <para>
/// The size in bytes.
/// </para>
/// <para></para>
/// </summary>
public static readonly long LinkIndexPartSizeInBytes =
    ↪ RawLinkIndexPart<TLinkAddress>.SizeInBytes;

/// <summary>
/// <para>
/// The size in bytes.
/// </para>
/// <para></para>
/// </summary>
public static readonly long LinkHeaderSizeInBytes =
    ↪ LinksHeader<TLinkAddress>.SizeInBytes;

/// <summary>
/// <para>
/// The default links size step.
/// </para>
/// <para></para>
/// </summary>
public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;

/// <summary>
/// <para>
/// The data memory.
/// </para>
/// <para></para>
/// </summary>
protected readonly IResizableDirectMemory _dataMemory;
/// <summary>
/// <para>
/// The index memory.
/// </para>
/// <para></para>
/// </summary>
protected readonly IResizableDirectMemory _indexMemory;
/// <summary>
/// <para>
/// The use linked list.
/// </para>
/// <para></para>
/// </summary>
protected readonly bool _useLinkedList;
/// <summary>
/// <para>
/// The data memory reservation step in bytes.
/// </para>
/// <para></para>
/// </summary>
protected readonly long _dataMemoryReservationStepInBytes;
/// <summary>
/// <para>
/// The index memory reservation step in bytes.
/// </para>
/// <para></para>
/// </summary>
protected readonly long _indexMemoryReservationStepInBytes;

/// <summary>
/// <para>
/// The internal sources list methods.
/// </para>

```

```

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

```

```

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

```

```

252 // Adjust reserved capacity
253 var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
254 if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
255 {
256     minimumDataReservedCapacity = dataMemory.UsedCapacity;
257 }
258 if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
259 {
260     minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
261 }
262 var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
263 if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
264 {
265     minimumIndexReservedCapacity = indexMemory.UsedCapacity;
266 }
267 if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
268 {
269     minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
270 }
271 // Check for alignment
272 if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
273 {
274     minimumDataReservedCapacity = ((minimumDataReservedCapacity /
275         ↪ _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
276         ↪ _dataMemoryReservationStepInBytes;
277 }
278 if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)
279 {
280     minimumIndexReservedCapacity = ((minimumIndexReservedCapacity /
281         ↪ _indexMemoryReservationStepInBytes) * _indexMemoryReservationStepInBytes) +
282         ↪ _indexMemoryReservationStepInBytes;
283 }
284 if (dataMemory.ReservedCapacity != minimumDataReservedCapacity)
285 {
286     dataMemory.ReservedCapacity = minimumDataReservedCapacity;
287 }
288 if (indexMemory.ReservedCapacity != minimumIndexReservedCapacity)
289 {
290     indexMemory.ReservedCapacity = minimumIndexReservedCapacity;
291 }
292 SetPointers(dataMemory, indexMemory);
293 header = ref GetHeaderReference();
294 // Ensure correctness _memory.UsedCapacity over _header->AllocatedLinks
295 // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
296 dataMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
297     ↪ LinkDataPartSizeInBytes) + LinkDataPartSizeInBytes; // First link is read only
298     ↪ zero link.
299 indexMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
300     ↪ LinkIndexPartSizeInBytes) + LinkHeaderSizeInBytes;
301 // Ensure correctness _memory.ReservedLinks over _header->ReservedCapacity
302 // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
303 header.ReservedLinks = ConvertToAddress((dataMemory.ReservedCapacity -
304     ↪ LinkDataPartSizeInBytes) / LinkDataPartSizeInBytes);
305 }
306
307 /// <summary>
308 /// <para>
309 /// Counts the substitution.
310 /// </para>
311 /// <para></para>
312 /// </summary>
313 /// <param name="restriction">
314 /// <para>The substitution.</para>
315 /// <para></para>
316 /// </param>
317 /// <exception cref="NotSupportedException">
318 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
319 /// <para></para>
320 /// </exception>
321 /// <returns>
322 /// <para>The link</para>
323 /// <para></para>
324 /// </returns>
325 [MethodImpl(MethodImplOptions.AggressiveInlining)]
326 public virtual TLinkAddress Count(ICollection<TLinkAddress>? restriction)
327 {
328     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
329     if (restriction.Count == 0)

```

```

322 {
323     return Total;
324 }
325 var constants = Constants;
326 var any = constants.Any;
327 var index = restriction[constants.IndexPart];
328 if (restriction.Count == 1)
329 {
330     if (AreEqual(index, any))
331     {
332         return Total;
333     }
334     return Exists(index) ? GetOne() : GetZero();
335 }
336 if (restriction.Count == 2)
337 {
338     var value = restriction[1];
339     if (AreEqual(index, any))
340     {
341         if (AreEqual(value, any))
342         {
343             return Total; // Any - как отсутствие ограничения
344         }
345         var externalReferencesRange = constants.ExternalReferencesRange;
346         if (externalReferencesRange.HasValue &&
347             ⇨ externalReferencesRange.Value.Contains(value))
348         {
349             return Add(ExternalSourcesTreeMethods.CountUsages(value),
350                 ⇨ ExternalTargetsTreeMethods.CountUsages(value));
351         }
352         else
353         {
354             if (_useLinkedList)
355             {
356                 return Add(InternalSourcesListMethods.CountUsages(value),
357                     ⇨ InternalTargetsTreeMethods.CountUsages(value));
358             }
359             else
360             {
361                 return Add(InternalSourcesTreeMethods.CountUsages(value),
362                     ⇨ InternalTargetsTreeMethods.CountUsages(value));
363             }
364         }
365     }
366     else
367     {
368         if (!Exists(index))
369         {
370             return GetZero();
371         }
372         if (AreEqual(value, any))
373         {
374             return GetOne();
375         }
376         ref var storedLinkValue = ref GetLinkDataPartReference(index);
377         if (AreEqual(storedLinkValue.Source, value) ||
378             ⇨ AreEqual(storedLinkValue.Target, value))
379         {
380             return GetOne();
381         }
382         return GetZero();
383     }
384 }
385 if (restriction.Count == 3)
386 {
387     var externalReferencesRange = constants.ExternalReferencesRange;
388     var source = restriction[constants.SourcePart];
389     var target = restriction[constants.TargetPart];
390     if (AreEqual(index, any))
391     {
392         if (AreEqual(source, any) && AreEqual(target, any))
393         {
394             return Total;
395         }
396         else if (AreEqual(source, any))
397         {
398             if (externalReferencesRange.HasValue &&
399                 ⇨ externalReferencesRange.Value.Contains(target))

```

```

394         {
395             return ExternalTargetsTreeMethods.CountUsages(target);
396         }
397         else
398         {
399             return InternalTargetsTreeMethods.CountUsages(target);
400         }
401     }
402     else if (AreEqual(target, any))
403     {
404         if (externalReferencesRange.HasValue &&
405             ↪ externalReferencesRange.Value.Contains(source))
406         {
407             return ExternalSourcesTreeMethods.CountUsages(source);
408         }
409         else
410         {
411             if (_useLinkedList)
412             {
413                 return InternalSourcesListMethods.CountUsages(source);
414             }
415             else
416             {
417                 return InternalSourcesTreeMethods.CountUsages(source);
418             }
419         }
420     }
421     else //if(source != Any && target != Any)
422     {
423         // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
424         TLinkAddress link;
425         if (externalReferencesRange.HasValue)
426         {
427             if (externalReferencesRange.Value.Contains(source) &&
428                 ↪ externalReferencesRange.Value.Contains(target))
429             {
430                 link = ExternalSourcesTreeMethods.Search(source, target);
431             }
432             else if (externalReferencesRange.Value.Contains(source))
433             {
434                 link = InternalTargetsTreeMethods.Search(source, target);
435             }
436             else if (externalReferencesRange.Value.Contains(target))
437             {
438                 if (_useLinkedList)
439                 {
440                     link = ExternalSourcesTreeMethods.Search(source, target);
441                 }
442                 else
443                 {
444                     link = InternalSourcesTreeMethods.Search(source, target);
445                 }
446             }
447             else
448             {
449                 if (_useLinkedList ||
450                     ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
451                     ↪ InternalTargetsTreeMethods.CountUsages(target)))
452                 {
453                     link = InternalTargetsTreeMethods.Search(source, target);
454                 }
455                 else
456                 {
457                     link = InternalSourcesTreeMethods.Search(source, target);
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     }
475 }

```



```

466     }
467     }
468     return AreEqual(link, constants.Null) ? GetZero() : GetOne();
469 }
470 }
471 else
472 {
473     if (!Exists(index))
474     {
475         return GetZero();
476     }
477     if (AreEqual(source, any) && AreEqual(target, any))
478     {
479         return GetOne();
480     }
481     ref var storedLinkValue = ref GetLinkDataPartReference(index);
482     if (!AreEqual(source, any) && !AreEqual(target, any))
483     {
484         if (AreEqual(storedLinkValue.Source, source) &&
485             ⇨ AreEqual(storedLinkValue.Target, target))
486         {
487             return GetOne();
488         }
489         return GetZero();
490     }
491     var value = default(TLinkAddress);
492     if (AreEqual(source, any))
493     {
494         value = target;
495     }
496     if (AreEqual(target, any))
497     {
498         value = source;
499     }
500     if (AreEqual(storedLinkValue.Source, value) ||
501         ⇨ AreEqual(storedLinkValue.Target, value))
502     {
503         return GetOne();
504     }
505     return GetZero();
506 }
507 }
508 }
509 throw new NotSupportedException("Другие размеры и способы ограничений не
510 ⇨ поддерживаются.");
511 }
512
513 /// <summary>
514 /// <para>
515 /// Eaches the handler.
516 /// </para>
517 /// <para></para>
518 /// </summary>
519 /// <param name="handler">
520 /// <para>The handler.</para>
521 /// <para></para>
522 /// </param>
523 /// <param name="restriction">
524 /// <para>The substitution.</para>
525 /// <para></para>
526 /// </param>
527 /// <exception cref="NotSupportedException">
528 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
529 /// <para></para>
530 /// </exception>
531 /// <returns>
532 /// <para>The link</para>
533 /// <para></para>
534 /// </returns>
535 [MethodImpl(MethodImplOptions.AggressiveInlining)]
536 public virtual TLinkAddress Each(ICollection<TLinkAddress>? restriction,
537 ⇨ ReadHandler<TLinkAddress>? handler)
538 {
539     var constants = Constants;
540     var @break = constants.Break;
541     if (restriction.Count == 0)
542     {
543         for (var link = GetOne(); LessOrEqualThan(link,
544             ⇨ GetHeaderReference().AllocatedLinks); link = Increment(link))

```

```

539     {
540         if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
541         {
542             return @break;
543         }
544     }
545     return @break;
546 }
547 var @continue = constants.Continue;
548 var any = constants.Any;
549 var index = restriction[constants.IndexPart];
550 if (restriction.Count == 1)
551 {
552     if (AreEqual(index, any))
553     {
554         return Each(Array.Empty<TLinkAddress>(), handler);
555     }
556     if (!Exists(index))
557     {
558         return @continue;
559     }
560     return handler(GetLinkStruct(index));
561 }
562 if (restriction.Count == 2)
563 {
564     var value = restriction[1];
565     if (AreEqual(index, any))
566     {
567         if (AreEqual(value, any))
568         {
569             return Each(Array.Empty<TLinkAddress>(), handler);
570         }
571         if (AreEqual(Each(new Link<TLinkAddress>(index, value, any), handler),
572             ↪ @break))
573         {
574             return @break;
575         }
576         return Each(new Link<TLinkAddress>(index, any, value), handler);
577     }
578     else
579     {
580         if (!Exists(index))
581         {
582             return @continue;
583         }
584         if (AreEqual(value, any))
585         {
586             return handler(GetLinkStruct(index));
587         }
588         ref var storedLinkValue = ref GetLinkDataPartReference(index);
589         if (AreEqual(storedLinkValue.Source, value) ||
590             AreEqual(storedLinkValue.Target, value))
591         {
592             return handler(GetLinkStruct(index));
593         }
594         return @continue;
595     }
596 }
597 if (restriction.Count == 3)
598 {
599     var externalReferencesRange = constants.ExternalReferencesRange;
600     var source = restriction[constants.SourcePart];
601     var target = restriction[constants.TargetPart];
602     if (AreEqual(index, any))
603     {
604         if (AreEqual(source, any) && AreEqual(target, any))
605         {
606             return Each(Array.Empty<TLinkAddress>(), handler);
607         }
608         else if (AreEqual(source, any))
609         {
610             if (externalReferencesRange.HasValue &&
611                 ↪ externalReferencesRange.Value.Contains(target))
612             {
613                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
614             }
615             else
616             {

```

```

615         return InternalTargetsTreeMethods.EachUsage(target, handler);
616     }
617 }
618 else if (AreEqual(target, any))
619 {
620     if (externalReferencesRange.HasValue &&
        ↪ externalReferencesRange.Value.Contains(source))
621     {
622         return ExternalSourcesTreeMethods.EachUsage(source, handler);
623     }
624     else
625     {
626         if (_useLinkedList)
627         {
628             return InternalSourcesListMethods.EachUsage(source, handler);
629         }
630         else
631         {
632             return InternalSourcesTreeMethods.EachUsage(source, handler);
633         }
634     }
635 }
636 else //if(source != Any && target != Any)
637 {
638     TLinkAddress link;
639     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 }

```

```

686     else
687     {
688         if (!Exists(index))
689         {
690             return @continue;
691         }
692         if (AreEqual(source, any) && AreEqual(target, any))
693         {
694             return handler(GetLinkStruct(index));
695         }
696         ref var storedLinkValue = ref GetLinkDataPartReference(index);
697         if (!AreEqual(source, any) && !AreEqual(target, any))
698         {
699             if (AreEqual(storedLinkValue.Source, source) &&
700                 AreEqual(storedLinkValue.Target, target))
701             {
702                 return handler(GetLinkStruct(index));
703             }
704             return @continue;
705         }
706         var value = default(TLinkAddress);
707         if (AreEqual(source, any))
708         {
709             value = target;
710         }
711         if (AreEqual(target, any))
712         {
713             value = source;
714         }
715         if (AreEqual(storedLinkValue.Source, value) ||
716             AreEqual(storedLinkValue.Target, value))
717         {
718             return handler(GetLinkStruct(index));
719         }
720         return @continue;
721     }
722 }
723 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↳ поддерживаются.");
724 }
725
726 /// <remarks>
727 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
728 ↳ в другом месте (но не в менеджере памяти, а в логике Links)
729 /// </remarks>
730 [MethodImpl(MethodImplOptions.AggressiveInlining)]
731 public virtual TLinkAddress Update(IList<TLinkAddress>? restriction,
732     ↳ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
733 {
734     var constants = Constants;
735     var @null = constants.Null;
736     var externalReferencesRange = constants.ExternalReferencesRange;
737     var linkIndex = restriction[constants.IndexPart];
738     var before = GetLinkStruct(linkIndex);
739     ref var link = ref GetLinkDataPartReference(linkIndex);
740     var source = link.Source;
741     var target = link.Target;
742     ref var header = ref GetHeaderReference();
743     ref var rootAsSource = ref header.RootAsSource;
744     ref var rootAsTarget = ref header.RootAsTarget;
745     // Будет корректно работать только в том случае, если пространство выделенной связи
746     ↳ предварительно заполнено нулями
747     if (!AreEqual(source, @null))
748     {
749         if (externalReferencesRange.HasValue &&
750             ↳ externalReferencesRange.Value.Contains(source))
751         {
752             ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
753         }
754         else
755         {
756             if (_useLinkedList)
757             {
758                 InternalSourcesListMethods.Detach(source, linkIndex);
759             }
760             else
761             {
762                 InternalSourcesTreeMethods.Detach(ref
763                     ↳ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
764             }
765         }
766     }
767 }

```

```

759     }
760 }
761 }
762 if (!AreEqual(target, @null))
763 {
764     if (externalReferencesRange.HasValue &&
765         ↪ externalReferencesRange.Value.Contains(target))
766     {
767         ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
768     }
769     else
770     {
771         InternalTargetsTreeMethods.Detach(ref
772             ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
773     }
774 }
775 source = link.Source = substitution[constants.SourcePart];
776 target = link.Target = substitution[constants.TargetPart];
777 if (!AreEqual(source, @null))
778 {
779     if (externalReferencesRange.HasValue &&
780         ↪ externalReferencesRange.Value.Contains(source))
781     {
782         ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
783     }
784     else
785     {
786         if (_useLinkedList)
787         {
788             InternalSourcesListMethods.AttachAsLast(source, linkIndex);
789         }
790         else
791         {
792             InternalSourcesTreeMethods.Attach(ref
793                 ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
794         }
795     }
796 }
797 if (!AreEqual(target, @null))
798 {
799     if (externalReferencesRange.HasValue &&
800         ↪ externalReferencesRange.Value.Contains(target))
801     {
802         ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
803     }
804     else
805     {
806         InternalTargetsTreeMethods.Attach(ref
807             ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
808     }
809 }
810 return handler?.Invoke(before, new Link<TLinkAddress>(linkIndex, source, target)) ??
811     ↪ Constants.Continue;
812 }
813
814 /// <remarks>
815 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
816 ↪ пространство
817 /// </remarks>
818 [MethodImpl(MethodImplOptions.AggressiveInlining)]
819 public virtual TLinkAddress Create(ICollection<TLinkAddress>? substitution,
820     ↪ WriteHandler<TLinkAddress>? handler)
821 {
822     ref var header = ref GetHeaderReference();
823     var freeLink = header.FirstFreeLink;
824     if (!AreEqual(freeLink, Constants.Null))
825     {
826         UnusedLinksListMethods.Detach(freeLink);
827     }
828     else
829     {
830         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
831         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
832         {
833             throw new
834                 ↪ LinksLimitReachedException<TLinkAddress>(maximumPossibleInnerReference);
835         }
836         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks))

```

```

827     {
828         _dataMemory.ReservedCapacity += _dataMemoryReservationStepInBytes;
829         _indexMemory.ReservedCapacity += _indexMemoryReservationStepInBytes;
830         SetPointers(_dataMemory, _indexMemory);
831         header = ref GetHeaderReference();
832         header.ReservedLinks = ConvertToAddress(_dataMemory.ReservedCapacity /
            ↳ LinkDataPartSizeInBytes);
833     }
834     freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
835     _dataMemory.UsedCapacity += LinkDataPartSizeInBytes;
836     _indexMemory.UsedCapacity += LinkIndexPartSizeInBytes;
837 }
838 return handler?.Invoke(null, GetLinkStruct(freeLink)) ?? Constants.Continue;
839 }
840
841 /// <summary>
842 /// <para>
843 /// Deletes the substitution.
844 /// </para>
845 /// <para></para>
846 /// </summary>
847 /// <param name="restriction">
848 /// <para>The substitution.</para>
849 /// <para></para>
850 /// </param>
851 [MethodImpl(MethodImplOptions.AggressiveInlining)]
852 public virtual 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,
867         // ↳ пока не дойдём до первой существующей связи
868         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
869         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
            ↳ IsUnusedLink(header.AllocatedLinks))
870         {
871             UnusedLinksListMethods.Detach(header.AllocatedLinks);
872             header.AllocatedLinks = Decrement(header.AllocatedLinks);
873             _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
874             _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
875         }
876     }
877     return handler?.Invoke(before, null) ?? Constants.Continue;
878 }
879
880 /// <summary>
881 /// <para>
882 /// Gets the link struct using the specified link index.
883 /// </para>
884 /// <para></para>
885 /// </summary>
886 /// <param name="linkIndex">
887 /// <para>The link index.</para>
888 /// <para></para>
889 /// </param>
890 /// <returns>
891 /// <para>A list of t link</para>
892 /// <para></para>
893 /// </returns>
894 [MethodImpl(MethodImplOptions.AggressiveInlining)]
895 public IList<TLinkAddress>? GetLinkStruct(TLinkAddress linkIndex)
896 {
897     ref var link = ref GetLinkDataPartReference(linkIndex);
898     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
899 }
900
901 /// <remarks>

```

```

901  /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
902  ↪ адрес реально поменялся
903  ///
904  /// Указатель this.links может быть в том же месте,
905  /// так как 0-я связь не используется и имеет такой же размер как Header,
906  /// поэтому header размещается в том же месте, что и 0-я связь
907  /// </remarks>
908  [MethodImpl(MethodImplOptions.AggressiveInlining)]
909  protected abstract void SetPointers(IResizableDirectMemory dataMemory,
910  ↪ IResizableDirectMemory indexMemory);
911
912  /// <summary>
913  /// <para>
914  /// Resets the pointers.
915  /// </para>
916  /// <para></para>
917  /// </summary>
918  [MethodImpl(MethodImplOptions.AggressiveInlining)]
919  protected virtual void ResetPointers()
920  {
921      InternalSourcesListMethods = null;
922      InternalSourcesTreeMethods = null;
923      ExternalSourcesTreeMethods = null;
924      InternalTargetsTreeMethods = null;
925      ExternalTargetsTreeMethods = null;
926      UnusedLinksListMethods = null;
927  }
928
929  /// <summary>
930  /// <para>
931  /// Gets the header reference.
932  /// </para>
933  /// <para></para>
934  /// </summary>
935  /// <returns>
936  /// <para>A ref links header of t link</para>
937  /// <para></para>
938  /// </returns>
939  [MethodImpl(MethodImplOptions.AggressiveInlining)]
940  protected abstract ref LinksHeader<TLinkAddress> GetHeaderReference();
941
942  /// <summary>
943  /// <para>
944  /// Gets the link data part reference using the specified link index.
945  /// </para>
946  /// <para></para>
947  /// </summary>
948  /// <param name="linkIndex">
949  /// <para>The link index.</para>
950  /// <para></para>
951  /// </param>
952  /// <returns>
953  /// <para>A ref raw link data part of t link</para>
954  /// <para></para>
955  /// </returns>
956  [MethodImpl(MethodImplOptions.AggressiveInlining)]
957  protected abstract ref RawLinkDataPart<TLinkAddress>
958  ↪ GetLinkDataPartReference(TLinkAddress linkIndex);
959
960  /// <summary>
961  /// <para>
962  /// Gets the link index part reference using the specified link index.
963  /// </para>
964  /// <para></para>
965  /// </summary>
966  /// <param name="linkIndex">
967  /// <para>The link index.</para>
968  /// <para></para>
969  /// </param>
970  /// <returns>
971  /// <para>A ref raw link index part of t link</para>
972  /// <para></para>
973  /// </returns>
974  [MethodImpl(MethodImplOptions.AggressiveInlining)]
975  protected abstract ref RawLinkIndexPart<TLinkAddress>
976  ↪ GetLinkIndexPartReference(TLinkAddress linkIndex);
977
978  /// <summary>

```

```

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

```



```

1050    /// <summary>
1051    /// <para>
1052    /// Determines whether this instance are equal.
1053    /// </para>
1054    /// <para></para>
1055    /// </summary>
1056    /// <param name="first">
1057    /// <para>The first.</para>
1058    /// <para></para>
1059    /// </param>
1060    /// <param name="second">
1061    /// <para>The second.</para>
1062    /// <para></para>
1063    /// </param>
1064    /// <returns>
1065    /// <para>The bool</para>
1066    /// <para></para>
1067    /// </returns>
1068    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1069    protected virtual bool AreEqual(TLinkAddress first, TLinkAddress second) =>
1070        ↪ _equalityComparer.Equals(first, second);
1071
1071    /// <summary>
1072    /// <para>
1073    /// 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
1092    /// <summary>
1093    /// <para>
1094    /// Determines whether this instance less or equal than.
1095    /// </para>
1096    /// <para></para>
1097    /// </summary>
1098    /// <param name="first">
1099    /// <para>The first.</para>
1100    /// <para></para>
1101    /// </param>
1102    /// <param name="second">
1103    /// <para>The second.</para>
1104    /// <para></para>
1105    /// </param>
1106    /// <returns>
1107    /// <para>The bool</para>
1108    /// <para></para>
1109    /// </returns>
1110    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1111    protected virtual bool LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
1112        ↪ _comparer.Compare(first, second) <= 0;
1113
1113    /// <summary>
1114    /// <para>
1115    /// Determines whether this instance greater than.
1116    /// </para>
1117    /// <para></para>
1118    /// </summary>
1119    /// <param name="first">
1120    /// <para>The first.</para>
1121    /// <para></para>
1122    /// </param>
1123    /// <param name="second">
1124    /// <para>The second.</para>

```

```

1125     /// <para></para>
1126     /// </param>
1127     /// <returns>
1128     /// <para>The bool</para>
1129     /// <para></para>
1130     /// </returns>
1131     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1132     protected virtual bool GreaterThan(TLinkAddress first, TLinkAddress second) =>
1133         ↪ _comparer.Compare(first, second) > 0;
1134
1135     /// <summary>
1136     /// <para>
1137     /// Determines whether this instance greater or equal than.
1138     /// </para>
1139     /// </summary>
1140     /// <param name="first">
1141     /// <para>The first.</para>
1142     /// <para></para>
1143     /// </param>
1144     /// <param name="second">
1145     /// <para>The second.</para>
1146     /// <para></para>
1147     /// </param>
1148     /// <returns>
1149     /// <para>The bool</para>
1150     /// <para></para>
1151     /// </returns>
1152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1153     protected virtual bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
1154         ↪ _comparer.Compare(first, second) >= 0;
1155
1156     /// <summary>
1157     /// <para>
1158     /// Converts the to int 64 using the specified value.
1159     /// </para>
1160     /// </summary>
1161     /// <param name="value">
1162     /// <para>The value.</para>
1163     /// <para></para>
1164     /// </param>
1165     /// <returns>
1166     /// <para>The long</para>
1167     /// <para></para>
1168     /// </returns>
1169     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1170     protected virtual long ConvertToInt64(TLinkAddress value) =>
1171         ↪ _addressToInt64Converter.Convert(value);
1172
1173     /// <summary>
1174     /// <para>
1175     /// Converts the to address using the specified value.
1176     /// </para>
1177     /// </summary>
1178     /// <param name="value">
1179     /// <para>The value.</para>
1180     /// <para></para>
1181     /// </param>
1182     /// <returns>
1183     /// <para>The link</para>
1184     /// <para></para>
1185     /// </returns>
1186     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1187     protected virtual TLinkAddress ConvertToAddress(long value) =>
1188         ↪ _int64ToAddressConverter.Convert(value);
1189
1190     /// <summary>
1191     /// <para>
1192     /// Adds the first.
1193     /// </para>
1194     /// <para></para>
1195     /// </summary>
1196     /// <param name="first">
1197     /// <para>The first.</para>
1198     /// <para></para>
1199     /// </param>

```

```

1199     /// <param name="second">
1200     /// <para>The second.</para>
1201     /// <para></para>
1202     /// </param>
1203     /// <returns>
1204     /// <para>The link</para>
1205     /// <para></para>
1206     /// </returns>
1207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1208     protected virtual TLinkAddress Add(TLinkAddress first, TLinkAddress second) =>
1209         ↪ Arithmetic<TLinkAddress>.Add(first, second);
1210
1211     /// <summary>
1212     /// <para>
1213     /// Subtracts the first.
1214     /// </para>
1215     /// <para></para>
1216     /// </summary>
1217     /// <param name="first">
1218     /// <para>The first.</para>
1219     /// <para></para>
1220     /// </param>
1221     /// <param name="second">
1222     /// <para>The second.</para>
1223     /// <para></para>
1224     /// </param>
1225     /// <returns>
1226     /// <para>The link</para>
1227     /// <para></para>
1228     /// </returns>
1229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1230     protected virtual TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
1231         ↪ Arithmetic<TLinkAddress>.Subtract(first, second);
1232
1233     /// <summary>
1234     /// <para>
1235     /// Increments the link.
1236     /// </para>
1237     /// <para></para>
1238     /// </summary>
1239     /// <param name="link">
1240     /// <para>The link.</para>
1241     /// <para></para>
1242     /// </param>
1243     /// <returns>
1244     /// <para>The link</para>
1245     /// <para></para>
1246     /// </returns>
1247     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1248     protected virtual TLinkAddress Increment(TLinkAddress link) =>
1249         ↪ Arithmetic<TLinkAddress>.Increment(link);
1250
1251     /// <summary>
1252     /// <para>
1253     /// Decrements the link.
1254     /// </para>
1255     /// <para></para>
1256     /// </summary>
1257     /// <param name="link">
1258     /// <para>The link.</para>
1259     /// <para></para>
1260     /// </param>
1261     /// <returns>
1262     /// <para>The link</para>
1263     /// <para></para>
1264     /// </returns>
1265     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1266     protected virtual TLinkAddress Decrement(TLinkAddress link) =>
1267         ↪ Arithmetic<TLinkAddress>.Decrement(link);
1268
1269     #region Disposable
1270
1271     /// <summary>
1272     /// <para>
1273     /// Gets the allow multiple dispose calls value.
1274     /// </para>
1275     /// <para></para>
1276     /// </summary>

```

```

1273     protected override bool AllowMultipleDisposeCalls
1274     {
1275         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1276         get => true;
1277     }
1278
1279     /// <summary>
1280     /// <para>
1281     /// Disposes the manual.
1282     /// </para>
1283     /// <para></para>
1284     /// </summary>
1285     /// <param name="manual">
1286     /// <para>The manual.</para>
1287     /// <para></para>
1288     /// </param>
1289     /// <param name="wasDisposed">
1290     /// <para>The was disposed.</para>
1291     /// <para></para>
1292     /// </param>
1293     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1294     protected override void Dispose(bool manual, bool wasDisposed)
1295     {
1296         if (!wasDisposed)
1297         {
1298             ResetPointers();
1299             _dataMemory.DisposeIfPossible();
1300             _indexMemory.DisposeIfPossible();
1301         }
1302     }
1303
1304     #endregion
1305 }
1306 }

```

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

```

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

```

```

42     _header = header;
43 }
44
45 /// <summary>
46 /// <para>
47 /// Gets the header reference.
48 /// </para>
49 /// <para></para>
50 /// </summary>
51 /// <returns>
52 /// <para>A ref links header of t link</para>
53 /// <para></para>
54 /// </returns>
55 [MethodImpl(MethodImplOptions.AggressiveInlining)]
56 protected virtual ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
    ↳ AsRef<LinksHeader<TLinkAddress>>(_header);
57
58 /// <summary>
59 /// <para>
60 /// Gets the link data part reference using the specified link.
61 /// </para>
62 /// <para></para>
63 /// </summary>
64 /// <param name="link">
65 /// <para>The link.</para>
66 /// <para></para>
67 /// </param>
68 /// <returns>
69 /// <para>A ref raw link data part of t link</para>
70 /// <para></para>
71 /// </returns>
72 [MethodImpl(MethodImplOptions.AggressiveInlining)]
73 protected virtual ref RawLinkDataPart<TLinkAddress>
    ↳ 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) =>
    ↳ GetHeaderReference().FirstFreeLink = element;
160
161 /// <summary>
162 /// <para>
163 /// Sets the last using the specified element.
164 /// </para>
165 /// <para></para>
166 /// </summary>
167 /// <param name="element">
168 /// <para>The element.</para>
169 /// <para></para>
170 /// </param>
171 [MethodImpl(MethodImplOptions.AggressiveInlining)]
172 protected override void SetLast(TLinkAddress element) =>
    ↳ GetHeaderReference().LastFreeLink = element;
173
174 /// <summary>
175 /// <para>
176 /// Sets the previous using the specified element.
177 /// </para>
178 /// <para></para>
179 /// </summary>
180 /// <param name="element">
181 /// <para>The element.</para>
182 /// <para></para>
183 /// </param>
184 /// <param name="previous">
185 /// <para>The previous.</para>
186 /// <para></para>
187 /// </param>
188 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

189     protected override void SetPrevious(TLinkAddress element, TLinkAddress previous) =>
190         ↪ GetLinkDataPartReference(element).Source = previous;
191
192     /// <summary>
193     /// <para>
194     /// Sets the next using the specified element.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="element">
199     /// <para>The element.</para>
200     /// <para></para>
201     /// </param>
202     /// <param name="next">
203     /// <para>The next.</para>
204     /// <para></para>
205     /// </param>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override void SetNext(TLinkAddress element, TLinkAddress next) =>
208         ↪ GetLinkDataPartReference(element).Target = next;
209
210     /// <summary>
211     /// <para>
212     /// Sets the size using the specified size.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     /// <param name="size">
217     /// <para>The size.</para>
218     /// <para></para>
219     /// </param>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     protected override void SetSize(TLinkAddress size) => GetHeaderReference().FreeLinks =
        ↪ size;
    }
}

```

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

```

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

```

```

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

```

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

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link index part.
13     /// </para>
14     /// <para></para>
15     /// </summary>

```



```

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

```

```

94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override bool Equals(object obj) => obj is RawLinkIndexPart<TLinkAddress> link ?
        ↳ Equals(link) : false;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance equals.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="other">
109    /// <para>The other.</para>
110    /// <para></para>
111    /// </param>
112    /// <returns>
113    /// <para>The bool</para>
114    /// <para></para>
115    /// </returns>
116    [MethodImpl(MethodImplOptions.AggressiveInlining)]
117    public bool Equals(RawLinkIndexPart<TLinkAddress> other)
118        => _equalityComparer.Equals(RootAsSource, other.RootAsSource)
119        && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
120        && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
121        && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
122        && _equalityComparer.Equals(RootAsTarget, other.RootAsTarget)
123        && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
124        && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
125        && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);
126
127    /// <summary>
128    /// <para>
129    /// Gets the hash code.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <returns>
134    /// <para>The int</para>
135    /// <para></para>
136    /// </returns>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    public override int GetHashCode() => (RootAsSource, LeftAsSource, RightAsSource,
        ↳ SizeAsSource, RootAsTarget, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
139
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    public static bool operator ==(RawLinkIndexPart<TLinkAddress> left,
        ↳ RawLinkIndexPart<TLinkAddress> right) => left.Equals(right);
142
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    public static bool operator !=(RawLinkIndexPart<TLinkAddress> left,
        ↳ RawLinkIndexPart<TLinkAddress> right) => !(left == right);
145 }
146 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLinkAddress = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 external links recursionless size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16     /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
17     public unsafe abstract class UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
        ↳ 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="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="constants">
49     /// <para>A constants.</para>
50     /// <para></para>
51     /// </param>
52     /// <param name="linksDataParts">
53     /// <para>A links data parts.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="linksIndexParts">
57     /// <para>A links index parts.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="header">
61     /// <para>A header.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
66     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
67     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
68     {
69         LinksDataParts = linksDataParts;
70         LinksIndexParts = linksIndexParts;
71         Header = header;
72     }
73
74     /// <summary>
75     /// <para>
76     /// Gets the zero.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <returns>
81     /// <para>The link</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override TLinkAddress GetZero() => 0U;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance equal to zero.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="value">
94     /// <para>The value.</para>

```

```

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) =>
181         ↪ first >= second;
182
183     /// <summary>
184     /// <para>
185     /// Determines whether this instance greater or equal than zero.
186     /// </para>
187     /// <para></para>
188     /// </summary>
189     /// <param name="value">
190     /// <para>The value.</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 GreaterOrEqualThanZero(TLinkAddress value) => true; // value >=
199         ↪ 0 is always true for ulong
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance less or equal than zero.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="value">
208     /// <para>The value.</para>
209     /// <para></para>
210     /// </param>
211     /// <returns>
212     /// <para>The bool</para>
213     /// <para></para>
214     /// </returns>
215     [MethodImpl(MethodImplOptions.AggressiveInlining)]
216     protected override bool LessOrEqualThanZero(TLinkAddress value) => value == OUL; //
217         ↪ value is always >= 0 for ulong
218
219     /// <summary>
220     /// <para>
221     /// Determines whether this instance less or equal than.
222     /// </para>
223     /// <para></para>
224     /// </summary>
225     /// <param name="first">
226     /// <para>The first.</para>
227     /// <para></para>
228     /// </param>
229     /// <param name="second">
230     /// <para>The second.</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 LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
239         ↪ first <= second;
240
241     /// <summary>
242     /// <para>
243     /// Determines whether this instance less than zero.
244     /// </para>
245     /// <para></para>
246     /// </summary>

```

```

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

```

```

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

```

```

394     /// </returns>
395     [MethodImpl(MethodImplOptions.AggressiveInlining)]
396     protected override ref RawLinkIndexPart<TLinkAddress>
        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
397
398     /// <summary>
399     /// <para>
400     /// Determines whether this instance first is to the left of second.
401     /// </para>
402     /// <para></para>
403     /// </summary>
404     /// <param name="first">
405     /// <para>The first.</para>
406     /// <para></para>
407     /// </param>
408     /// <param name="second">
409     /// <para>The second.</para>
410     /// <para></para>
411     /// </param>
412     /// <returns>
413     /// <para>The bool</para>
414     /// <para></para>
415     /// </returns>
416     [MethodImpl(MethodImplOptions.AggressiveInlining)]
417     protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
418     {
419         ref var firstLink = ref LinksDataParts[first];
420         ref var secondLink = ref LinksDataParts[second];
421         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
422     }
423
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance first is to the right of second.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The bool</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
        ↪ second)
444     {
445         ref var firstLink = ref LinksDataParts[first];
446         ref var secondLink = ref LinksDataParts[second];
447         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
448     }
449 }
450 }

```

## 1.51 ./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"/>
    ↳ instance.
44      /// </para>
45      /// <para></para>
46      /// </summary>
47      /// <param name="constants">
48      /// <para>A constants.</para>
49      /// <para></para>
50      /// </param>
51      /// <param name="linksDataParts">
52      /// <para>A links data parts.</para>
53      /// <para></para>
54      /// </param>
55      /// <param name="linksIndexParts">
56      /// <para>A links index parts.</para>
57      /// <para></para>
58      /// </param>
59      /// <param name="header">
60      /// <para>A header.</para>
61      /// <para></para>
62      /// </param>
63      [MethodImpl(MethodImplOptions.AggressiveInlining)]
64      protected UInt32ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
    ↳ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65      {
66          LinksDataParts = linksDataParts;
67          LinksIndexParts = linksIndexParts;
68          Header = header;
69      }
70
71      /// <summary>
72      /// <para>
73      /// Gets the zero.
74      /// </para>
75      /// <para></para>
76      /// </summary>
77      /// <returns>
78      /// <para>The link</para>
79      /// <para></para>
80      /// </returns>
81      [MethodImpl(MethodImplOptions.AggressiveInlining)]
82      protected override TLinkAddress GetZero() => 0U;
83
84      /// <summary>
85      /// <para>
86      /// Determines whether this instance equal to zero.
87      /// </para>
88      /// <para></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) =>
181         ↪ first >= second;
182
183     /// <summary>
184     /// <para>
185     /// <para>Determines whether this instance greater or equal than zero.
186     /// </para>
187     /// <para></para>
188     /// </summary>
189     /// <param name="value">
190     /// <para>The value.</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 GreaterOrEqualThanZero(TLinkAddress value) => true; // value >=
199         ↪ 0 is always true for ulong
200
201     /// <summary>
202     /// <para>
203     /// <para>Determines whether this instance less or equal than zero.
204     /// </para>
205     /// <para></para>
206     /// </summary>
207     /// <param name="value">
208     /// <para>The value.</para>
209     /// <para></para>
210     /// </param>
211     /// <returns>
212     /// <para>The bool</para>
213     /// <para></para>
214     /// </returns>
215     [MethodImpl(MethodImplOptions.AggressiveInlining)]
216     protected override bool LessOrEqualThanZero(TLinkAddress value) => value == OUL; //
217         ↪ value is always >= 0 for ulong
218
219     /// <summary>
220     /// <para>
221     /// <para>Determines whether this instance less or equal than.
222     /// </para>
223     /// <para></para>
224     /// </summary>
225     /// <param name="first">
226     /// <para>The first.</para>
227     /// <para></para>
228     /// </param>
229     /// <param name="second">
230     /// <para>The second.</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 LessOrEqualThan(TLinkAddress first, TLinkAddress second) =>
239         ↪ first <= second;
240
241     /// <summary>
242     /// <para>
243     /// <para>Determines whether this instance less than zero.

```

```

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

```

```

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

```

```

391     /// <returns>
392     /// <para>A ref raw link index part of t link</para>
393     /// <para></para>
394     /// </returns>
395     [MethodImpl(MethodImplOptions.AggressiveInlining)]
396     protected override ref RawLinkIndexPart<TLinkAddress>
        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
397
398     /// <summary>
399     /// <para>
400     /// Determines whether this instance first is to the left of second.
401     /// </para>
402     /// <para></para>
403     /// </summary>
404     /// <param name="first">
405     /// <para>The first.</para>
406     /// <para></para>
407     /// </param>
408     /// <param name="second">
409     /// <para>The second.</para>
410     /// <para></para>
411     /// </param>
412     /// <returns>
413     /// <para>The bool</para>
414     /// <para></para>
415     /// </returns>
416     [MethodImpl(MethodImplOptions.AggressiveInlining)]
417     protected override bool 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,
            ↪ secondLink.Source, secondLink.Target);
422     }
423
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance first is to the right of second.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The bool</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
        ↪ second)
444     {
445         ref var firstLink = ref LinksDataParts[first];
446         ref var secondLink = ref LinksDataParts[second];
447         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
448     }
449 }
450 }

```

## 1.52 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalanced

```

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 :
    ↳ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16  {
17      /// <summary>
18      /// <para>
19      /// Initializes a new <see
    ↳ cref="UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
20      /// </para>
21      /// <para></para>
22      /// </summary>
23      /// <param name="constants">
24      /// <para>A constants.</para>
25      /// <para></para>
26      /// </param>
27      /// <param name="linksDataParts">
28      /// <para>A links data parts.</para>
29      /// <para></para>
30      /// </param>
31      /// <param name="linksIndexParts">
32      /// <para>A links index parts.</para>
33      /// <para></para>
34      /// </param>
35      /// <param name="header">
36      /// <para>A header.</para>
37      /// <para></para>
38      /// </param>
39      [MethodImpl(MethodImplOptions.AggressiveInlining)]
40      public UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<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>

```



```

158 [MethodImpl(MethodImplOptions.AggressiveInlining)]
159 protected override TLinkAddress GetSize(TLinkAddress node) =>
    ↳ LinksIndexParts[node].SizeAsSource;

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

177
178 /// <summary>
179 /// <para>
180 /// Gets the tree root.
181 /// </para>
182 /// <para></para>
183 /// </summary>
184 /// <returns>
185 /// <para>The link</para>
186 /// <para></para>
187 /// </returns>
188 [MethodImpl(MethodImplOptions.AggressiveInlining)]
189 protected override TLinkAddress GetTreeRoot() => Header->RootAsSource;
190
191 /// <summary>
192 /// <para>
193 /// Gets the base part value using the specified node.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <param name="node">
198 /// <para>The node.</para>
199 /// <para></para>
200 /// </param>
201 /// <returns>
202 /// <para>The link</para>
203 /// <para></para>
204 /// </returns>
205 [MethodImpl(MethodImplOptions.AggressiveInlining)]
206 protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
    ↳ LinksDataParts[node].Source;

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

```

```

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

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

266     /// <summary>
267     /// <para>
268     /// Clears the node using the specified node.
269     /// </para>
270     /// <para></para>
271     /// </summary>
272     /// <param name="node">
273     /// <para>The node.</para>
274     /// <para></para>
275     /// </param>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void ClearNode(TLinkAddress node)
278     {
279         ref var link = ref LinksIndexParts[node];
280         link.LeftAsSource = Zero;
281         link.RightAsSource = Zero;
282         link.SizeAsSource = Zero;
283     }
284 }
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) =>
92         ↪ LinksIndexParts[node].LeftAsSource;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLinkAddress GetRight(TLinkAddress node) =>
109        ↪ LinksIndexParts[node].RightAsSource;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="node">
118    /// <para>The node.</para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// </param>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
125        ↪ 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    /// </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     /// <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         ↳ LinksIndexParts[node].SizeAsSource = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <returns>
186     /// <para>The link</para>
187     /// <para></para>
188     /// </returns>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override TLinkAddress GetTreeRoot() => Header->RootAsSource;
191
192     /// <summary>
193     /// <para>
194     /// Gets the base part value using the specified node.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="node">
199     /// <para>The node.</para>
200     /// <para></para>
201     /// </param>
202     /// <returns>
203     /// <para>The link</para>
204     /// <para></para>
205     /// </returns>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
208         ↳ LinksDataParts[node].Source;
209
210     /// <summary>
211     /// <para>
212     /// Determines whether this instance first is to the left of second.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     /// <param name="firstSource">
217     /// <para>The first source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="firstTarget">
221     /// <para>The first target.</para>
222     /// <para></para>
223     /// </param>
224     /// <param name="secondSource">
225     /// <para>The second source.</para>
226     /// <para></para>
227     /// </param>
228     /// <param name="secondTarget">
229     /// <para>The second target.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
238         ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)

```

```

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

```

#### 1.54 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/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     /// <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        ↪ 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>

```



```

166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="size">
172     /// <para>The size.</para>
173     /// <para></para>
174     /// </param>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
177         ↳ LinksIndexParts[node].SizeAsTarget = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root.
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>
197     /// </summary>
198     /// <param name="node">
199     /// <para>The node.</para>
200     /// <para></para>
201     /// </param>
202     /// <returns>
203     /// <para>The link</para>
204     /// <para></para>
205     /// </returns>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
208         ↳ LinksDataParts[node].Target;
209
210     /// <summary>
211     /// <para>
212     /// Determines whether this instance first is to the left of second.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     /// <param name="firstSource">
217     /// <para>The first source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="firstTarget">
221     /// <para>The first target.</para>
222     /// <para></para>
223     /// </param>
224     /// <param name="secondSource">
225     /// <para>The second source.</para>
226     /// <para></para>
227     /// </param>
228     /// <param name="secondTarget">
229     /// <para>The second target.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
238         ↳ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
239         => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
240             ↳ secondSource;
241
242     /// <summary>
243     /// <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     /// <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         ref var link = ref LinksIndexParts[node];
281         link.LeftAsTarget = Zero;
282         link.RightAsTarget = Zero;
283         link.SizeAsTarget = Zero;
284     }
285 }
286 }
287 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksTargetsSizeBalancedTreeMethods :
        ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32ExternalLinksTargetsSizeBalancedTreeMethods"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>

```

```

25     /// <para></para>
26     /// </param>
27     /// <param name="linksDataParts">
28     /// <para>A links data parts.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt32 ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        ↪ : base(constants, linksDataParts, linksIndexParts, header) { }

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

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

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

92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>

```

```

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

```

```

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>
197     /// </summary>
198     /// <param name="node">
199     /// <para>The node.</para>
200     /// <para></para>
201     /// </param>
202     /// <returns>
203     /// <para>The link</para>
204     /// <para></para>
205     /// </returns>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
208         ↪ LinksDataParts[node].Target;
209
210     /// <summary>
211     /// <para>
212     /// Determines whether this instance first is to the left of second.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     /// <param name="firstSource">
217     /// <para>The first source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="firstTarget">
221     /// <para>The first target.</para>
222     /// <para></para>
223     /// </param>
224     /// <param name="secondSource">
225     /// <para>The second source.</para>
226     /// <para></para>
227     /// </param>
228     /// <param name="secondTarget">
229     /// <para>The second target.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress firstSource, TLinkAddress
238         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget)
239         => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
240         ↪ secondSource;
241
242     /// <summary>
243     /// <para>
244     /// Determines whether this instance first is to the right of second.
245     /// </para>
246     /// <para></para>
247     /// </summary>
248     /// <param name="firstSource">

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLinkAddress = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 internal links recursionless size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16     public unsafe abstract class UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
        ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
17     {
18         /// <summary>
19         /// <para>
20         /// The links data parts.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
25         /// <summary>
26         /// <para>
27         /// The links index parts.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;

```

```

32     /// <summary>
33     /// <para>
34     /// The header.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     protected new readonly LinksHeader<TLinkAddress>* Header;
39
40     /// <summary>
41     /// <para>
42     /// Initializes a new <see
43     ↪ cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="constants">
48     /// <para>A constants.</para>
49     /// </param>
50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// </param>
53     /// <param name="linksIndexParts">
54     /// <para>A links index parts.</para>
55     /// </param>
56     /// <param name="header">
57     /// <para>A header.</para>
58     /// </param>
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     protected UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
61     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
62     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
63     {
64         LinksDataParts = linksDataParts;
65         LinksIndexParts = linksIndexParts;
66         Header = header;
67     }
68
69     /// <summary>
70     /// <para>
71     /// Gets the zero.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <returns>
76     /// <para>The link</para>
77     /// <para></para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     protected override TLinkAddress GetZero() => 0U;
81
82     /// <summary>
83     /// <para>
84     /// Determines whether this instance equal to zero.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="value">
89     /// <para>The value.</para>
90     /// </param>
91     /// <returns>
92     /// <para>The bool</para>
93     /// <para></para>
94     /// </returns>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     protected override bool EqualToZero(TLinkAddress value) => value == 0U;
97
98     /// <summary>
99     /// <para>
100     /// Determines whether this instance are equal.
101     /// </para>
102     /// <para></para>
103     /// </summary>

```

```

107    /// <param name="first">
108    /// <para>The first.</para>
109    /// <para></para>
110    /// </param>
111    /// <param name="second">
112    /// <para>The second.</para>
113    /// <para></para>
114    /// </param>
115    /// <returns>
116    /// <para>The bool</para>
117    /// <para></para>
118    /// </returns>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    protected override bool AreEqual(TLinkAddress first, TLinkAddress second) => first ==
    ↪ second;

121
122    /// <summary>
123    /// <para>
124    /// Determines whether this instance greater than zero.
125    /// </para>
126    /// <para></para>
127    /// </summary>
128    /// <param name="value">
129    /// <para>The value.</para>
130    /// <para></para>
131    /// </param>
132    /// <returns>
133    /// <para>The bool</para>
134    /// <para></para>
135    /// </returns>
136    [MethodImpl(MethodImplOptions.AggressiveInlining)]
137    protected override bool GreaterThanZero(TLinkAddress value) => value > 0U;

138
139    /// <summary>
140    /// <para>
141    /// Determines whether this instance greater than.
142    /// </para>
143    /// <para></para>
144    /// </summary>
145    /// <param name="first">
146    /// <para>The first.</para>
147    /// <para></para>
148    /// </param>
149    /// <param name="second">
150    /// <para>The second.</para>
151    /// <para></para>
152    /// </param>
153    /// <returns>
154    /// <para>The bool</para>
155    /// <para></para>
156    /// </returns>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override bool GreaterThan(TLinkAddress first, TLinkAddress second) => first >
    ↪ second;

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

180
181    /// <summary>

```



```

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

```

```

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

```

```

332    /// </para>
333    /// <para></para>
334    /// </summary>
335    /// <param name="first">
336    /// <para>The first.</para>
337    /// <para></para>
338    /// </param>
339    /// <param name="second">
340    /// <para>The second.</para>
341    /// <para></para>
342    /// </param>
343    /// <returns>
344    /// <para>The link</para>
345    /// <para></para>
346    /// </returns>
347    [MethodImpl(MethodImplOptions.AggressiveInlining)]
348    protected override TLinkAddress Subtract(TLinkAddress first, TLinkAddress second) =>
349        ↪ first - second;
350
351    /// <summary>
352    /// <para>
353    /// Gets the link data part reference using the specified link.
354    /// </para>
355    /// <para></para>
356    /// </summary>
357    /// <param name="link">
358    /// <para>The link.</para>
359    /// <para></para>
360    /// </param>
361    /// <returns>
362    /// <para>A ref raw link data part of t link</para>
363    /// <para></para>
364    /// </returns>
365    [MethodImpl(MethodImplOptions.AggressiveInlining)]
366    protected override ref RawLinkDataPart<TLinkAddress>
367        ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
368
369    /// <summary>
370    /// <para>
371    /// Gets the link index part reference using the specified link.
372    /// </para>
373    /// <para></para>
374    /// </summary>
375    /// <param name="link">
376    /// <para>The link.</para>
377    /// <para></para>
378    /// </param>
379    /// <returns>
380    /// <para>A ref raw link index part of t link</para>
381    /// <para></para>
382    /// </returns>
383    [MethodImpl(MethodImplOptions.AggressiveInlining)]
384    protected override ref RawLinkIndexPart<TLinkAddress>
385        ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
386
387    /// <summary>
388    /// <para>
389    /// Determines whether this instance first is to the left of second.
390    /// </para>
391    /// <para></para>
392    /// </summary>
393    /// <param name="first">
394    /// <para>The first.</para>
395    /// <para></para>
396    /// </param>
397    /// <param name="second">
398    /// <para>The second.</para>
399    /// <para></para>
400    /// </param>
401    /// <returns>
402    /// <para>The bool</para>
403    /// <para></para>
404    /// </returns>
405    [MethodImpl(MethodImplOptions.AggressiveInlining)]
406    protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress
407        ↪ second) => GetKeyPartValue(first) < GetKeyPartValue(second);
408
409    /// <summary>

```

```

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

```

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

```

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

```

```

53     /// </param>
54     /// <param name="linksIndexParts">
55     /// <para>A links index parts.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="header">
59     /// <para>A header.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64     {
65         LinksDataParts = linksDataParts;
66         LinksIndexParts = linksIndexParts;
67         Header = header;
68     }
69
70
71     /// <summary>
72     /// <para>
73     /// Gets the zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <returns>
78     /// <para>The link</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override TLinkAddress GetZero() => OU;
83
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance equal to zero.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override bool EqualToZero(TLinkAddress value) => value == OU;
100
101     /// <summary>
102     /// <para>
103     /// Determines whether this instance are equal.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="first">
108     /// <para>The first.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="second">
112     /// <para>The second.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The bool</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override bool AreEqual(TLinkAddress first, TLinkAddress second) => first ==
        ↪ second;
121
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 >
159    ↪ 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) =>
181    ↪ first >= second;
182
183    /// <summary>
184    /// <para>
185    /// Determines whether this instance greater or equal than zero.
186    /// </para>
187    /// <para></para>
188    /// </summary>
189    /// <param name="value">
190    /// <para>The value.</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 GreaterOrEqualThanZero(TLinkAddress value) => true; // value >=
199    ↪ 0 is always true for ulong
200
201    /// <summary>
202    /// <para>
203    /// Determines whether this instance less or equal than zero.
204    /// </para>
205    /// <para></para>
206    /// </summary>

```

```

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

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 :
16         ↪ InternalLinksSourcesLinkedListMethods<TLinkAddress>
17     {
18         private readonly RawLinkDataPart<TLinkAddress>* _linksDataParts;
19         private readonly RawLinkIndexPart<TLinkAddress>* _linksIndexParts;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt32InternalLinksSourcesLinkedListMethods"/> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="constants">
28         /// <para>A constants.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksDataParts">
32         /// <para>A links data parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="linksIndexParts">
36         /// <para>A links index parts.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress>
41             ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
42             ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts)
43             : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
44         {
45             _linksDataParts = linksDataParts;
46             _linksIndexParts = linksIndexParts;
47         }
48
49         /// <summary>
50         /// <para>
51         /// Gets the link data part reference using the specified link.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         /// <param name="link">
56         /// <para>The link.</para>
57         /// <para></para>
58         /// </param>
59         /// <returns>
60         /// <para>A ref raw link data part of t link</para>
61         /// <para></para>
62         /// </returns>
63         [MethodImpl(MethodImplOptions.AggressiveInlining)]
64         protected override ref RawLinkDataPart<TLinkAddress>
65             ↪ GetLinkDataPartReference(TLinkAddress link) => ref _linksDataParts[link];
66
67         /// <summary>
68         /// <para>
69         /// Gets the link index 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 index part of t link</para>
```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
        ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
        ↪ 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<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
41
42         /// <summary>
43         /// <para>
44         /// Gets the left reference using the specified node.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="node">
49         /// <para>The node.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>The ref link</para>
54         /// <para></para>
55         /// </returns>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
        ↪ LinksIndexParts[node].LeftAsSource;
58
59         /// <summary>
60         /// <para>
61         /// Gets the right reference using the specified node.
62         /// </para>
63         /// <para></para>

```

```

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

```

```

138    /// <para>The right.</para>
139    /// <para></para>
140    /// </param>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
143        ↪ LinksIndexParts[node].RightAsSource = right;
144
145    /// <summary>
146    /// <para>
147    /// Gets the size using the specified node.
148    /// </para>
149    /// </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    /// </summary>
165    /// <param name="node">
166    /// <para>The node.</para>
167    /// </param>
168    /// <param name="size">
169    /// <para>The size.</para>
170    /// </param>
171    [MethodImpl(MethodImplOptions.AggressiveInlining)]
172    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
173        ↪ LinksIndexParts[node].SizeAsSource = size;
174
175    /// <summary>
176    /// <para>
177    /// Gets the tree root using the specified node.
178    /// </para>
179    /// </summary>
180    /// <param name="node">
181    /// <para>The node.</para>
182    /// </param>
183    /// <returns>
184    /// <para>The link</para>
185    /// </returns>
186    [MethodImpl(MethodImplOptions.AggressiveInlining)]
187    protected override TLinkAddress GetTreeRoot(TLinkAddress node) =>
188        ↪ LinksIndexParts[node].RootAsSource;
189
190    /// <summary>
191    /// <para>
192    /// Gets the base part value using the specified node.
193    /// </para>
194    /// </summary>
195    /// <param name="node">
196    /// <para>The node.</para>
197    /// </param>
198    /// <returns>
199    /// <para>The link</para>
200    /// </returns>
201    [MethodImpl(MethodImplOptions.AggressiveInlining)]
202    protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
203        ↪ LinksDataParts[node].Source;

```

```

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

```

## 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

```

```

16 {
17     /// <summary>
18     /// <para>
19     /// Initializes a new <see cref="UInt32InternalLinksSourcesSizeBalancedTreeMethods"/>
20     ↪ instance.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     /// <param name="constants">
25     /// <para>A constants.</para>
26     /// <para></para>
27     /// </param>
28     /// <param name="linksDataParts">
29     /// <para>A links data parts.</para>
30     /// <para></para>
31     /// </param>
32     /// <param name="linksIndexParts">
33     /// <para>A links index parts.</para>
34     /// <para></para>
35     /// </param>
36     /// <param name="header">
37     /// <para>A header.</para>
38     /// <para></para>
39     /// </param>
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     public UInt32InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
42     ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
43     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
44     ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
45
46     /// <summary>
47     /// <para>
48     /// Gets the left reference using the specified node.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="node">
53     /// <para>The node.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>The ref link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
62     ↪ LinksIndexParts[node].LeftAsSource;
63
64     /// <summary>
65     /// <para>
66     /// Gets the right reference using the specified node.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="node">
71     /// <para>The node.</para>
72     /// <para></para>
73     /// </param>
74     /// <returns>
75     /// <para>The ref link</para>
76     /// <para></para>
77     /// </returns>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
80     ↪ LinksIndexParts[node].RightAsSource;
81
82     /// <summary>
83     /// <para>
84     /// Gets the left using the specified node.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="node">
89     /// <para>The node.</para>
90     /// <para></para>
91     /// </param>
92     /// <returns>
93     /// <para>The link</para>

```

```

88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLinkAddress GetLeft(TLinkAddress node) =>
92         ↪ LinksIndexParts[node].LeftAsSource;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLinkAddress GetRight(TLinkAddress node) =>
109        ↪ LinksIndexParts[node].RightAsSource;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="node">
118    /// <para>The node.</para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// </param>
123    [MethodImpl(MethodImplOptions.AggressiveInlining)]
124    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
125        ↪ 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    /// </param>
136    /// <param name="right">
137    /// <para>The right.</para>
138    /// </param>
139    [MethodImpl(MethodImplOptions.AggressiveInlining)]
140    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
141        ↪ LinksIndexParts[node].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    /// </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        ↪ LinksIndexParts[node].SizeAsSource;

```



```

160
161     /// <summary>
162     /// <para>
163     /// Sets the size using the specified node.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="size">
172     /// <para>The size.</para>
173     /// <para></para>
174     /// </param>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
177         ↪ LinksIndexParts[node].SizeAsSource = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root using the specified node.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="node">
186     /// <para>The node.</para>
187     /// <para></para>
188     /// </param>
189     /// <returns>
190     /// <para>The link</para>
191     /// <para></para>
192     /// </returns>
193     [MethodImpl(MethodImplOptions.AggressiveInlining)]
194     protected override TLinkAddress GetTreeRoot(TLinkAddress node) =>
195         ↪ LinksIndexParts[node].RootAsSource;
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].Source;
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].Target;
232
233     /// <summary>
234     /// <para>
235     /// Clears the node using the specified node.
236     /// </para>
237     /// <para></para>

```

```

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

```

## 1.61 ./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>

```

```

39 [MethodImpl(MethodImplOptions.AggressiveInlining)]
40 public UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
    ↳ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    ↳ : base(constants, linksDataParts, linksIndexParts, header) { }

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

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

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

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

```

```

110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
126        ↪ LinksIndexParts[node].LeftAsTarget = left;
127
128    /// <summary>
129    /// <para>
130    /// Sets the right using the specified node.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="node">
135    /// <para>The node.</para>
136    /// <para></para>
137    /// </param>
138    /// <param name="right">
139    /// <para>The right.</para>
140    /// <para></para>
141    /// </param>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
144        ↪ LinksIndexParts[node].RightAsTarget = right;
145
146    /// <summary>
147    /// <para>
148    /// Gets the size using the specified node.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="node">
153    /// <para>The node.</para>
154    /// <para></para>
155    /// </param>
156    /// <returns>
157    /// <para>The link</para>
158    /// <para></para>
159    /// </returns>
160    [MethodImpl(MethodImplOptions.AggressiveInlining)]
161    protected override TLinkAddress GetSize(TLinkAddress node) =>
162        ↪ LinksIndexParts[node].SizeAsTarget;
163
164    /// <summary>
165    /// <para>
166    /// Sets the size using the specified node.
167    /// </para>
168    /// <para></para>
169    /// </summary>
170    /// <param name="node">
171    /// <para>The node.</para>
172    /// <para></para>
173    /// </param>
174    /// <param name="size">
175    /// <para>The size.</para>
176    /// <para></para>
177    /// </param>
178    [MethodImpl(MethodImplOptions.AggressiveInlining)]
179    protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
180        ↪ LinksIndexParts[node].SizeAsTarget = size;
181
182    /// <summary>
183    /// <para>
184    /// Gets the tree root using the specified node.
185    /// </para>
186    /// <para></para>
187    /// </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].RootAsTarget;
195
196    /// <summary>
197    /// <para>
198    /// Gets the base part value using the specified node.
199    /// </para>
200    /// <para></para>
201    /// </summary>
202    /// <param name="node">
203    /// <para>The node.</para>
204    /// <para></para>
205    /// </param>
206    /// <returns>
207    /// <para>The link</para>
208    /// <para></para>
209    /// </returns>
210    [MethodImpl(MethodImplOptions.AggressiveInlining)]
211    protected override TLinkAddress GetBasePartValue(TLinkAddress node) =>
212        ↪ LinksDataParts[node].Target;
213
214    /// <summary>
215    /// <para>
216    /// Gets the key part value using the specified node.
217    /// </para>
218    /// <para></para>
219    /// </summary>
220    /// <param name="node">
221    /// <para>The node.</para>
222    /// <para></para>
223    /// </param>
224    /// <returns>
225    /// <para>The link</para>
226    /// <para></para>
227    /// </returns>
228    [MethodImpl(MethodImplOptions.AggressiveInlining)]
229    protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
230        ↪ LinksDataParts[node].Source;
231
232    /// <summary>
233    /// <para>
234    /// Clears the node using the specified node.
235    /// </para>
236    /// <para></para>
237    /// </summary>
238    /// <param name="node">
239    /// <para>The node.</para>
240    /// <para></para>
241    /// </param>
242    [MethodImpl(MethodImplOptions.AggressiveInlining)]
243    protected override void ClearNode(TLinkAddress node)
244    {
245        ref var link = ref LinksIndexParts[node];
246        link.LeftAsTarget = Zero;
247        link.RightAsTarget = Zero;
248        link.SizeAsTarget = Zero;
249    }
250
251    /// <summary>
252    /// <para>
253    /// Searches the source.
254    /// </para>
255    /// <para></para>
256    /// </summary>
257    /// <param name="source">
258    /// <para>The source.</para>
259    /// <para></para>
260    /// </param>
261    /// <param name="target">

```

```

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

```

## 1.62 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/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"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UInt32InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
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
        ↪ LinksIndexParts[node].LeftAsTarget;
59
60         /// <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) =>
    ↪ LinksIndexParts[node].LeftAsTarget = left;
126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="node">

```

```

134    /// <para>The node.</para>
135    /// <para></para>
136    /// </param>
137    /// <param name="right">
138    /// <para>The right.</para>
139    /// <para></para>
140    /// </param>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
143        ↪ LinksIndexParts[node].RightAsTarget = right;
144
145    /// <summary>
146    /// <para>
147    /// Gets the size using the specified node.
148    /// </para>
149    /// <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>
207    /// <para></para>
208    /// </param>
209    /// <returns>
210    /// <para>The link</para>
211    /// <para></para>
212    /// </returns>

```



```

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

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLinkAddress = System.UInt32;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>

```

```

14  /// Represents the int 32 split memory links.
15  /// </para>
16  /// <para></para>
17  /// </summary>
18  /// <seealso cref="SplitMemoryLinksBase{TLinkAddress}"/>
19  public unsafe class UInt32SplitMemoryLinks : SplitMemoryLinksBase<TLinkAddress>
20  {
21      private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalSourceTreeMethods;
22      private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalSourceTreeMethods;
23      private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalTargetTreeMethods;
24      private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalTargetTreeMethods;
25      private LinksHeader<TLinkAddress>* _header;
26      private RawLinkDataPart<TLinkAddress>* _linksDataParts;
27      private RawLinkIndexPart<TLinkAddress>* _linksIndexParts;
28
29      /// <summary>
30      /// <para>
31      /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
32      /// </para>
33      /// <para></para>
34      /// </summary>
35      /// <param name="dataMemory">
36      /// <para>A data memory.</para>
37      /// <para></para>
38      /// </param>
39      /// <param name="indexMemory">
40      /// <para>A index memory.</para>
41      /// <para></para>
42      /// </param>
43      [MethodImpl(MethodImplOptions.AggressiveInlining)]
44      public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
45      ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
46
47      /// <summary>
48      /// <para>
49      /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
50      /// </para>
51      /// <para></para>
52      /// </summary>
53      /// <param name="dataMemory">
54      /// <para>A data memory.</para>
55      /// <para></para>
56      /// </param>
57      /// <param name="indexMemory">
58      /// <para>A index memory.</para>
59      /// <para></para>
60      /// </param>
61      /// <param name="memoryReservationStep">
62      /// <para>A memory reservation step.</para>
63      /// <para></para>
64      /// </param>
65      [MethodImpl(MethodImplOptions.AggressiveInlining)]
66      public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
67      ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
68      ↪ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
69      ↪ IndexTreeType.Default, useLinkedList: true) { }
70
71      /// <summary>
72      /// <para>
73      /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
74      /// </para>
75      /// <para></para>
76      /// </summary>
77      /// <param name="dataMemory">
78      /// <para>A data memory.</para>
79      /// <para></para>
80      /// </param>
81      /// <param name="indexMemory">
82      /// <para>A index memory.</para>
83      /// <para></para>
84      /// </param>
85      /// <param name="memoryReservationStep">
86      /// <para>A memory reservation step.</para>
87      /// <para></para>
88      /// </param>
89      /// <param name="constants">
90      /// <para>A constants.</para>
91      /// <para></para>
92      /// </param>

```

```

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,
158     ↪ IResizableDirectMemory indexMemory)
159 {
160     _linksDataParts = (RawLinkDataPart<TLinkAddress>*)dataMemory.Pointer;
161     _linksIndexParts = (RawLinkIndexPart<TLinkAddress>*)indexMemory.Pointer;
162     _header = (LinksHeader<TLinkAddress>*)indexMemory.Pointer;
163     if (_useLinkedList)
164     {
165         InternalSourcesListMethods = new
166             ↪ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
167             ↪ _linksIndexParts);
168     }
169     else
170     {
171         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
172     }
173     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
174     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
175     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
176     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
177 }
178
179 /// <summary>
180 /// <para>
181 /// Resets the pointers.
182 /// </para>
183 /// <para></para>
184 /// </summary>
185 [MethodImpl(MethodImplOptions.AggressiveInlining)]
186 protected override void ResetPointers()
187 {
188     base.ResetPointers();
189     _linksDataParts = null;
190     _linksIndexParts = null;
191     _header = null;
192 }
193
194 /// <summary>
195 /// <para>
196 /// Gets the header reference.
197 /// </para>
198 /// <para></para>
199 /// </summary>
200 /// <returns>
201 /// <para>A ref links header of t link</para>
202 /// <para></para>
203 /// </returns>
204 [MethodImpl(MethodImplOptions.AggressiveInlining)]
205 protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref *_header;
206
207 /// <summary>
208 /// <para>
209 /// Gets the link data part reference using the specified link index.
210 /// </para>
211 /// <para></para>
212 /// </summary>
213 /// <param name="linkIndex">
214 /// <para>The link index.</para>
215 /// <para></para>
216 /// </param>
217 /// <returns>
218 /// <para>A ref raw link data part of t link</para>
219 /// <para></para>
220 /// </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) =>
300         ↪ first <= second;
301
302     /// <summary>
303     /// <para>
304     /// Determines whether this instance greater than.
305     /// </para>
306     /// <para></para>
307     /// </summary>
308     /// <param name="first">
309     /// <para>The first.</para>
310     /// <para></para>
311     /// </param>
312     /// <param name="second">
313     /// <para>The second.</para>
314     /// <para></para>
315     /// </param>
316     /// <returns>
317     /// <para>The bool</para>
318     /// <para></para>
319     /// </returns>
320     [MethodImpl(MethodImplOptions.AggressiveInlining)]
321     protected override bool GreaterThan(TLinkAddress first, TLinkAddress second) => first >
322         ↪ second;
323
324     /// <summary>
325     /// <para>
326     /// Determines whether this instance greater or equal than.
327     /// </para>
328     /// <para></para>
329     /// </summary>
330     /// <param name="first">
331     /// <para>The first.</para>
332     /// <para></para>
333     /// </param>
334     /// <param name="second">
335     /// <para>The second.</para>
336     /// <para></para>
337     /// </param>
338     /// <returns>
339     /// <para>The bool</para>
340     /// <para></para>
341     /// </returns>
342     [MethodImpl(MethodImplOptions.AggressiveInlining)]
343     protected override bool GreaterOrEqualThan(TLinkAddress first, TLinkAddress second) =>
344         ↪ first >= second;
345
346     /// <summary>
347     /// <para>
348     /// Gets the zero.
349     /// </para>
350     /// <para></para>
351     /// </summary>
352     /// <returns>
353     /// <para>The link</para>
354     /// <para></para>
355     /// </returns>
356     [MethodImpl(MethodImplOptions.AggressiveInlining)]
357     protected override TLinkAddress GetZero() => OU;
358
359     /// <summary>
360     /// <para>
361     /// Gets the one.
362     /// </para>
363     /// <para></para>
364     /// </summary>
365     /// <returns>
366     /// <para>The link</para>
367     /// <para></para>
368     /// </returns>
369     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLinkAddress = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 unused links list methods.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="UnusedLinksListMethods{TLinkAddress}"/>
16     public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<TLinkAddress>
17     {
18         private readonly RawLinkDataPart<TLinkAddress>* _links;
19         private readonly LinksHeader<TLinkAddress>* _header;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt32UnusedLinksListMethods(RawLinkDataPart<TLinkAddress>* links,
37             ↪ LinksHeader<TLinkAddress>* header)
38             : base((byte*)links, (byte*)header)
39         {

```



```

39         _links = links;
40         _header = header;
41     }
42
43     /// <summary>
44     /// <para>
45     /// Gets the link data part reference using the specified link.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="link">
50     /// <para>The link.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>A ref raw link data part of t link</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     protected override ref RawLinkDataPart<TLinkAddress>
59     ↪ GetLinkDataPartReference(TLinkAddress link) => ref _links[link];
60
61     /// <summary>
62     /// <para>
63     /// Gets the header reference.
64     /// </para>
65     /// <para></para>
66     /// </summary>
67     /// <returns>
68     /// <para>A ref links header of t link</para>
69     /// <para></para>
70     /// </returns>
71     [MethodImpl(MethodImplOptions.AggressiveInlining)]
72     protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref *_header;
73 }

```

## 1.65 ./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 :
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;

```

```

40
41 /// <summary>
42 /// <para>
43 /// Initializes a new <see
44   ↳ cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45 /// </para>
46 /// <para></para>
47 /// </summary>
48 /// <param name="constants">
49 /// <para>A constants.</para>
50 /// <para></para>
51 /// </param>
52 /// <param name="linksDataParts">
53 /// <para>A links data parts.</para>
54 /// <para></para>
55 /// </param>
56 /// <param name="linksIndexParts">
57 /// <para>A links index parts.</para>
58 /// <para></para>
59 /// </param>
60 /// <param name="header">
61 /// <para>A header.</para>
62 /// <para></para>
63 /// </param>
64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
protected UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLi
   ↳ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
   ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
: base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
{
    LinksDataParts = linksDataParts;
    LinksIndexParts = linksIndexParts;
    Header = header;
}

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)]
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)]
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>
380    ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
381
382    /// <summary>
383    /// <para>
384    /// Gets the link index part reference using the specified link.
385    /// </para>
386    /// <para></para>
387    /// </summary>
388    /// <param name="link">
389    /// <para>The link.</para>
390    /// <para></para>
391    /// </param>
392    /// <returns>
393    /// <para>A ref raw link index part of t link</para>
394    /// <para></para>
395    /// </returns>
396    [MethodImpl(MethodImplOptions.AggressiveInlining)]
397    protected override ref RawLinkIndexPart<TLinkAddress>
398    ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
399
400    /// <summary>
401    /// <para>
402    /// Determines whether this instance first is to the left of second.
403    /// </para>
404    /// <para></para>
405    /// </summary>
406    /// <param name="first">
407    /// <para>The first.</para>
408    /// <para></para>
409    /// </param>
410    /// <param name="second">
411    /// <para>The second.</para>
412    /// <para></para>
413    /// </param>
414    /// <returns>
415    /// <para>The bool</para>
416    /// <para></para>
417    /// </returns>
418    [MethodImpl(MethodImplOptions.AggressiveInlining)]
419    protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
420    {
421        ref var firstLink = ref LinksDataParts[first];
422        ref var secondLink = ref LinksDataParts[second];

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLinkAddress = System.UInt64;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 64 external links size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16    /// <seealso cref="ILinksTreeMethods{TLinkAddress}"/>
17    public unsafe abstract class UInt64ExternalLinksSizeBalancedTreeMethodsBase :
18        ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
19    {
20        /// <summary>
21        /// <para>
22        /// The links data parts.
23        /// </para>
24        /// <para></para>
25        /// </summary>
26        protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
27        /// <summary>
28        /// <para>
29        /// The links index parts.
30        /// </para>
31        /// <para></para>
32        /// </summary>
33        protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
34        /// <summary>
35        /// <para>
36        /// The header.
37        /// </para>
38        /// <para></para>
39        /// </summary>
40        protected new readonly LinksHeader<TLinkAddress>* Header;
41
42        /// <summary>
43        /// <para>

```

```

43     /// Initializes a new <see cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
44     ↪ instance.
45     /// </para>
46     /// </summary>
47     /// <param name="constants">
48     /// <para>A constants.</para>
49     /// </param>
50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// </param>
53     /// <param name="linksIndexParts">
54     /// <para>A links index parts.</para>
55     /// </param>
56     /// <param name="header">
57     /// <para>A header.</para>
58     /// </param>
59     /// <param name="header">
60     /// <para>A header.</para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected UInt64ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress>
64     ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
65     ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
66     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
67     {
68         LinksDataParts = linksDataParts;
69         LinksIndexParts = linksIndexParts;
70         Header = header;
71     }
72     /// <summary>
73     /// <para>
74     /// Gets the zero.
75     /// </para>
76     /// </summary>
77     /// <returns>
78     /// <para>The ulong</para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     protected override ulong GetZero() => OUL;
82     /// <summary>
83     /// <para>
84     /// Determines whether this instance equal to zero.
85     /// </para>
86     /// </summary>
87     /// <param name="value">
88     /// <para>The value.</para>
89     /// </param>
90     /// <returns>
91     /// <para>The bool</para>
92     /// </returns>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     protected override bool EqualToZero(ulong value) => value == OUL;
95     /// <summary>
96     /// <para>
97     /// Determines whether this instance are equal.
98     /// </para>
99     /// </summary>
100    /// <param name="first">
101    /// <para>The first.</para>
102    /// </param>
103    /// <param name="second">
104    /// <para>The second.</para>
105    /// </param>
106    /// <returns>
107    /// <para>The bool</para>

```



```

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

```

```

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

```

```

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

```

```

349     protected override ulong Subtract(ulong first, ulong second) => first - second;
350
351     /// <summary>
352     /// <para>
353     /// Gets the header reference.
354     /// </para>
355     /// <para></para>
356     /// </summary>
357     /// <returns>
358     /// <para>A ref links header of t link</para>
359     /// <para></para>
360     /// </returns>
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref *Header;
363
364     /// <summary>
365     /// <para>
366     /// Gets the link data part reference using the specified link.
367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="link">
371     /// <para>The link.</para>
372     /// <para></para>
373     /// </param>
374     /// <returns>
375     /// <para>A ref raw link data part of t link</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     protected override ref RawLinkDataPart<TLinkAddress>
380     ↪ GetLinkDataPartReference(TLinkAddress link) => ref LinksDataParts[link];
381
382     /// <summary>
383     /// <para>
384     /// Gets the link index part reference using the specified link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>A ref raw link index part of t link</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override ref RawLinkIndexPart<TLinkAddress>
398     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
399
400     /// <summary>
401     /// <para>
402     /// Determines whether this instance first is to the left of second.
403     /// </para>
404     /// <para></para>
405     /// </summary>
406     /// <param name="first">
407     /// <para>The first.</para>
408     /// <para></para>
409     /// </param>
410     /// <param name="second">
411     /// <para>The second.</para>
412     /// <para></para>
413     /// </param>
414     /// <returns>
415     /// <para>The bool</para>
416     /// <para></para>
417     /// </returns>
418     [MethodImpl(MethodImplOptions.AggressiveInlining)]
419     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
420     {
421         ref var firstLink = ref LinksDataParts[first];
422         ref var secondLink = ref LinksDataParts[second];
423         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
424             ↪ secondLink.Source, secondLink.Target);
425     }

```

```

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/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethodsBase

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 external links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
    ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
    ↪ cref="UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
    ↪ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
41
42     /// <summary>

```

```

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

```

```

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 FirstIsToLeftOfSecond(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)]

```



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

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

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, LinkHeader<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></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
58     ↪ LinksIndexParts[node].LeftAsSource;
59
60     /// <summary>
61     /// <para>
62     /// Gets the right reference using the specified node.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="node">
67     /// <para>The node.</para>
68     /// <para></para>
69     /// </param>
70     /// <returns>
71     /// <para>The ref link</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
76     ↪ LinksIndexParts[node].RightAsSource;
77
78     /// <summary>
79     /// <para>
80     /// Gets the left using the specified node.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="node">
85     /// <para>The node.</para>
86     /// <para></para>
87     /// </param>
88     /// <returns>
89     /// <para>The link</para>
90     /// <para></para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     protected override TLinkAddress GetLeft(TLinkAddress node) =>
94     ↪ LinksIndexParts[node].LeftAsSource;
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].RightAsSource;
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>

```

```

122 /// <para></para>
123 /// </param>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
126     ↳ LinksIndexParts[node].LeftAsSource = left;
127
128 /// <summary>
129 /// <para>
130 /// Sets the right using the specified node.
131 /// </para>
132 /// <para></para>
133 /// </summary>
134 /// <param name="node">
135 /// <para>The node.</para>
136 /// </param>
137 /// <param name="right">
138 /// <para>The right.</para>
139 /// <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 /// </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 /// </param>
171 /// <param name="size">
172 /// <para>The size.</para>
173 /// <para></para>
174 /// </param>
175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
176 protected override void SetSize(TLinkAddress node, TLinkAddress size) =>
177     ↳ LinksIndexParts[node].SizeAsSource = size;
178
179 /// <summary>
180 /// <para>
181 /// Gets the tree root.
182 /// </para>
183 /// <para></para>
184 /// </summary>
185 /// <returns>
186 /// <para>The link</para>
187 /// <para></para>
188 /// </returns>
189 [MethodImpl(MethodImplOptions.AggressiveInlining)]
190 protected override TLinkAddress GetTreeRoot() => Header->RootAsSource;
191
192 /// <summary>
193 /// <para>
194 /// Gets the base part value using the specified node.
195 /// </para>
196 /// <para></para>

```

```

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     /// <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         => firstSource < secondSource || firstSource == secondSource && firstTarget <
239         ↪ secondTarget;
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         => firstSource > secondSource || firstSource == secondSource && firstTarget >
271         ↪ secondTarget;

```

```

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

```

## 1.69 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 external links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16     ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<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>

```

```

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) =>
    ↪ LinksIndexParts[node].LeftAsTarget = left;

```

```

126
127     /// <summary>
128     /// <para>
129     /// Sets the right using the specified node.
130     /// </para>
131     /// <para></para>
132     /// </summary>
133     /// <param name="node">
134     /// <para>The node.</para>
135     /// <para></para>
136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
143         ↪ LinksIndexParts[node].RightAsTarget = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// <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.
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->RootAsTarget;
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>

```

```

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

```



```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using TLinkAddress = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 external links targets size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt64ExternalLinksTargetsSizeBalancedTreeMethods :
16    ↪ UInt64ExternalLinksSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see cref="UInt64ExternalLinksTargetsSizeBalancedTreeMethods"/>
21        ↪ instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public UInt64ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
43        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
44        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
45        ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
46
47        /// <summary>
48        /// <para>
49        /// Gets the left reference using the specified node.
50        /// </para>
51        /// <para></para>
52        /// </summary>
53        /// <param name="node">
54        /// <para>The node.</para>
55        /// <para></para>
56        /// </param>
57        /// <returns>
58        /// <para>The ref link</para>
59        /// <para></para>
60        /// </returns>
61        [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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.
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->RootAsTarget;
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>

```

```

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

```

```

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/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.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 internal links recursionless size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLinkAddress}"/>
16     public unsafe abstract class UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17     ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
18     {
19         /// <summary>
20         /// <para>
21         /// The links data parts.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLinkDataPart<TLinkAddress>* LinksDataParts;
26
27         /// <summary>
28         /// <para>
29         /// The links index parts.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected new readonly RawLinkIndexPart<TLinkAddress>* LinksIndexParts;
34
35         /// <summary>
36         /// <para>
37         /// The header.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         protected new readonly LinksHeader<TLinkAddress>* Header;
42
43         /// <summary>
44         /// <para>
45         /// Initializes a new <see
46         ↪ cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="constants">
51         /// <para>A constants.</para>
52         /// <para></para>
53         /// </param>
54         /// <param name="linksDataParts">
55         /// <para>A links data parts.</para>
56         /// <para></para>
57         /// </param>
58         /// <param name="linksIndexParts">
59         /// <para>A links index parts.</para>
60         /// <para></para>
61         /// </param>
62         /// <param name="header">
63         /// <para>A header.</para>
64         /// <para></para>
65         /// </param>
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

63     protected UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
64         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65     {
66         LinksDataParts = linksDataParts;
67         LinksIndexParts = linksIndexParts;
68         Header = header;
69     }
70
71     /// <summary>
72     /// <para>
73     /// Gets the zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <returns>
78     /// <para>The ulong</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override ulong GetZero() => OUL;
83
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance equal to zero.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override bool EqualToZero(ulong value) => value == OUL;
100
101     /// <summary>
102     /// <para>
103     /// Determines whether this instance are equal.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="first">
108     /// <para>The first.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="second">
112     /// <para>The second.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The bool</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override bool AreEqual(ulong first, ulong second) => first == second;
121
122     /// <summary>
123     /// <para>
124     /// Determines whether this instance greater than zero.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="value">
129     /// <para>The value.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The bool</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected override bool GreaterThanZero(ulong value) => value > OUL;
138

```

```

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

```

```

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

```



```

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>

```

```

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>
383     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref LinksIndexParts[link];
384
385     /// <summary>
386     /// <para>
387     /// Determines whether this instance first is to the left of second.
388     /// </para>
389     /// <para></para>
390     /// </summary>
391     /// <param name="first">
392     /// <para>The first.</para>
393     /// <para></para>
394     /// </param>
395     /// <param name="second">
396     /// <para>The second.</para>
397     /// <para></para>
398     /// </param>
399     /// <returns>
400     /// <para>The bool</para>
401     /// <para></para>
402     /// </returns>
403     [MethodImpl(MethodImplOptions.AggressiveInlining)]
404     protected override bool FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress
405     ↪ second) => GetKeyPartValue(first) < GetKeyPartValue(second);
406
407     /// <summary>
408     /// <para>
409     /// Determines whether this instance first is to the right of second.
410     /// </para>
411     /// <para></para>
412     /// </summary>
413     /// <param name="first">
414     /// <para>The first.</para>
415     /// <para></para>
416     /// </param>
417     /// <param name="second">
418     /// <para>The second.</para>
419     /// <para></para>
420     /// </param>
421     /// <returns>
422     /// <para>The bool</para>
423     /// <para></para>
424     /// </returns>
425     [MethodImpl(MethodImplOptions.AggressiveInlining)]
426     protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
427     ↪ second) => GetKeyPartValue(first) > GetKeyPartValue(second);
428 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using 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"/>
    ↳ 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 UInt64InternalLinksSizeBalancedTreeMethodsBase(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     /// <summary>
71     /// <para>
72     /// Gets the zero.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <returns>
77     /// <para>The ulong</para>
78     /// <para></para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     protected override ulong GetZero() => 0UL;
82
83     /// <summary>
84     /// <para>
85     /// Determines whether this instance equal to zero.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89

```

```

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

```

```

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

```

```

244     /// <para></para>
245     /// </param>
246     /// <returns>
247     /// <para>The bool</para>
248     /// <para></para>
249     /// </returns>
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪     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>A bool</para>
402     /// <para></para>
403     /// </returns>

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using TLinkAddress = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Generic
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 internal links sources linked list methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLinkAddress}"/>
15    public unsafe class UInt64InternalLinksSourcesLinkedListMethods :
    ↪ InternalLinksSourcesLinkedListMethods<TLinkAddress>
16    {
17        private readonly RawLinkDataPart<TLinkAddress>* _linksDataParts;
18        private readonly RawLinkIndexPart<TLinkAddress>* _linksIndexParts;
19
20        /// <summary>
21        /// <para>
22        /// Initializes a new <see cref="UInt64InternalLinksSourcesLinkedListMethods"/> instance.
23        /// </para>
24        /// <para></para>
25        /// </summary>
26        /// <param name="constants">
27        /// <para>A constants.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="linksDataParts">
31        /// <para>A links data parts.</para>
32        /// <para></para>
33        /// </param>
34        /// <param name="linksIndexParts">
35        /// <para>A links index parts.</para>
36        /// <para></para>
37        /// </param>
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        public UInt64InternalLinksSourcesLinkedListMethods(LinksConstants<TLinkAddress>
    ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts)
    : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
40    {
41

```



```

42         _linksDataParts = linksDataParts;
43         _linksIndexParts = linksIndexParts;
44     }
45
46     /// <summary>
47     /// <para>
48     /// Gets the link data part reference using the specified link.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="link">
53     /// <para>The link.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>A ref raw link data part of t link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref RawLinkDataPart<TLinkAddress>
62     ↪ GetLinkDataPartReference(TLinkAddress link) => ref _linksDataParts[link];
63
64     /// <summary>
65     /// <para>
66     /// Gets the link index part reference using the specified link.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="link">
71     /// <para>The link.</para>
72     /// <para></para>
73     /// </param>
74     /// <returns>
75     /// <para>A ref raw link index part of t link</para>
76     /// <para></para>
77     /// </returns>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected override ref RawLinkIndexPart<TLinkAddress>
80     ↪ GetLinkIndexPartReference(TLinkAddress link) => ref _linksIndexParts[link];
81 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
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>

```

```

34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt64 InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
    ↪ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
    ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
    ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
41
42     /// <summary>
43     /// <para>
44     /// Gets the left reference using the specified node.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="node">
49     /// <para>The node.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
    ↪ LinksIndexParts[node].LeftAsSource;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↪ LinksIndexParts[node].RightAsSource;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLinkAddress GetLeft(TLinkAddress node) =>
    ↪ LinksIndexParts[node].LeftAsSource;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>

```

```

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

```

```

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) =>
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>

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links sources size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksSourcesSizeBalancedTreeMethods :
        ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64InternalLinksSourcesSizeBalancedTreeMethods"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UInt64InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLinkAddress>
        ↪ constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
        ↪ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
        ↪ : base(constants, linksDataParts, linksIndexParts, header) { }
41
42         /// <summary>
43         /// <para>
44         /// Gets the left reference using the specified node.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="node">
49         /// <para>The node.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>The ref link</para>
54         /// <para></para>

```

```

55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
    ↪ LinksIndexParts[node].LeftAsSource;

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

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

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

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

```

```

127     /// <summary>
128     /// <para>
129     /// Sets the right using the specified node.
130     /// </para>
131     /// <para></para>
132     /// </summary>
133     /// <param name="node">
134     /// <para>The node.</para>
135     /// <para></para>
136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(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>
204     /// </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 }
271 }

```

## 1.76 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalance

```

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

```



```

6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 internal links targets recursionless size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16        ↳ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see
21        ↳ cref="UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLi
43        ↳ nkAddress> constants, RawLinkDataPart<TLinkAddress>* linksDataParts,
44        ↳ RawLinkIndexPart<TLinkAddress>* linksIndexParts, LinksHeader<TLinkAddress>* header)
45        ↳ : base(constants, linksDataParts, linksIndexParts, header) { }
46
47        /// <summary>
48        /// <para>
49        /// Gets the left reference using the specified node.
50        /// </para>
51        /// <para></para>
52        /// </summary>
53        /// <param name="node">
54        /// <para>The node.</para>
55        /// <para></para>
56        /// </param>
57        /// <returns>
58        /// <para>The ref ulong</para>
59        /// <para></para>
60        /// </returns>
61        [MethodImpl(MethodImplOptions.AggressiveInlining)]
62        protected override ref ulong GetLeftReference(ulong 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 ulong</para>
77        /// <para></para>
78        /// </returns>
79        [MethodImpl(MethodImplOptions.AggressiveInlining)]
80        protected override ref ulong GetRightReference(ulong node) => ref
81        ↳ LinksIndexParts[node].RightAsTarget;
82
83        /// <summary>

```

```

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

```

```

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

```

```

225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected override TLinkAddress GetKeyPartValue(TLinkAddress node) =>
228         ↪ LinksDataParts[node].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);

```

## 1.77 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods

```

1  using System.Runtime.CompilerServices;
2  using TLinkAddress = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksTargetsSizeBalancedTreeMethods :
16         ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
21         /// instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>

```

```

29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public 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) =>
109        ↪ LinksIndexParts[node].RightAsTarget;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></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>

```

```

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) =>
    ↳ LinksDataParts[node].Target;

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

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

```

```

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

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLinkAddress = System.UInt64;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the int 64 split memory links.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="SplitMemoryLinksBase{TLinkAddress}"/>
19     public unsafe class UInt64SplitMemoryLinks : SplitMemoryLinksBase<TLinkAddress>
20     {
21         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalSourceTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createInternalTargetTreeMethods;
24         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createExternalTargetTreeMethods;
25         private LinksHeader<ulong>* _header;
26         private RawLinkDataPart<ulong>* _linksDataParts;
27         private RawLinkIndexPart<ulong>* _linksIndexParts;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         /// <param name="dataMemory">
36         /// <para>A data memory.</para>
37         /// <para></para>
38         /// </param>
39         /// <param name="indexMemory">
40         /// <para>A index memory.</para>
41         /// <para></para>
42         /// </param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46         /// <summary>
47         /// <para>
48         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <param name="dataMemory">
53         /// <para>A data memory.</para>
54         /// <para></para>
55         /// </param>

```



```

56     /// <param name="indexMemory">
57     /// <para>A index memory.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="memoryReservationStep">
61     /// <para>A memory reservation step.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↳ memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
        ↳ IndexTreeType.Default, useLinkedList: true) { }

66     /// <summary>
67     /// <para>
68     /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
69     /// </para>
70     /// <para></para>
71     /// </summary>
72     /// <param name="dataMemory">
73     /// <para>A data memory.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="indexMemory">
77     /// <para>A index memory.</para>
78     /// <para></para>
79     /// </param>
80     /// <param name="memoryReservationStep">
81     /// <para>A memory reservation step.</para>
82     /// <para></para>
83     /// </param>
84     /// <param name="constants">
85     /// <para>A constants.</para>
86     /// <para></para>
87     /// </param>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep, LinksConstants<TLinkAddress> constants) :
        ↳ this(dataMemory, indexMemory, memoryReservationStep, constants,
        ↳ IndexTreeType.Default, useLinkedList: true) { }

91     /// <summary>
92     /// <para>
93     /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <param name="dataMemory">
98     /// <para>A data memory.</para>
99     /// <para></para>
100    /// </param>
101    /// <param name="indexMemory">
102    /// <para>A index memory.</para>
103    /// <para></para>
104    /// </param>
105    /// <param name="memoryReservationStep">
106    /// <para>A memory reservation step.</para>
107    /// <para></para>
108    /// </param>
109    /// <param name="constants">
110    /// <para>A constants.</para>
111    /// <para></para>
112    /// </param>
113    /// <param name="indexTreeType">
114    /// <para>A index tree type.</para>
115    /// <para></para>
116    /// </param>
117    /// <param name="useLinkedList">
118    /// <para>A use linked list.</para>
119    /// <para></para>
120    /// </param>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    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
128             ↳ UInt64InternalLinksSourcesSizeBalancedTreeMethods(Constants,
129                 ↳ _linksDataParts, _linksIndexParts, _header);
130         _createExternalSourceTreeMethods = () => new
131             ↳ UInt64ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
132                 ↳ _linksDataParts, _linksIndexParts, _header);
133         _createInternalTargetTreeMethods = () => new
134             ↳ UInt64InternalLinksTargetsSizeBalancedTreeMethods(Constants,
135                 ↳ _linksDataParts, _linksIndexParts, _header);
136         _createExternalTargetTreeMethods = () => new
137             ↳ UInt64ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
138                 ↳ _linksDataParts, _linksIndexParts, _header);
139     }
140     else
141     {
142         _createInternalSourceTreeMethods = () => new
143             ↳ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
144                 ↳ _linksDataParts, _linksIndexParts, _header);
145         _createExternalSourceTreeMethods = () => new
146             ↳ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
147                 ↳ _linksDataParts, _linksIndexParts, _header);
148         _createInternalTargetTreeMethods = () => new
149             ↳ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
150                 ↳ _linksDataParts, _linksIndexParts, _header);
151         _createExternalTargetTreeMethods = () => new
152             ↳ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
153                 ↳ _linksDataParts, _linksIndexParts, _header);
154     }
155     Init(dataMemory, indexMemory);
156 }
157
158 /// <summary>
159 /// <para>
160 /// Sets the pointers using the specified data memory.
161 /// </para>
162 /// <para></para>
163 /// </summary>
164 /// <param name="dataMemory">
165 /// <para>The data memory.</para>
166 /// <para></para>
167 /// </param>
168 /// <param name="indexMemory">
169 /// <para>The index memory.</para>
170 /// <para></para>
171 /// </param>
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 protected override void SetPointers(IResizableDirectMemory dataMemory,
174     ↳ IResizableDirectMemory indexMemory)
175 {
176     _linksDataParts = (RawLinkDataPart<TLinkAddress>*)dataMemory.Pointer;
177     _linksIndexParts = (RawLinkIndexPart<TLinkAddress>*)indexMemory.Pointer;
178     _header = (LinksHeader<TLinkAddress>*)indexMemory.Pointer;
179     if (_useLinkedList)
180     {
181         InternalSourcesListMethods = new
182             ↳ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
183                 ↳ _linksIndexParts);
184     }
185     else
186     {
187         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
188     }
189     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
190     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
191     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
192     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_linksDataParts, _header);
193 }
194
195 /// <summary>
196 /// <para>
197 /// Resets the pointers.
198 /// </para>
199 /// <para></para>
200 /// </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>
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;

```

```

258     /// <summary>
259     /// <para>
260     /// Determines whether this instance less than.
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 LessThan(ulong first, ulong second) => first < second;
278
279     /// <summary>
280     /// <para>
281     /// Determines whether this instance less or equal than.
282     /// </para>
283     /// <para></para>
284     /// </summary>
285     /// <param name="first">
286     /// <para>The first.</para>
287     /// <para></para>
288     /// </param>
289     /// <param name="second">
290     /// <para>The second.</para>
291     /// <para></para>
292     /// </param>
293     /// <returns>
294     /// <para>The bool</para>
295     /// <para></para>
296     /// </returns>
297     [MethodImpl(MethodImplOptions.AggressiveInlining)]
298     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
299
300     /// <summary>
301     /// <para>
302     /// Determines whether this instance greater than.
303     /// </para>
304     /// <para></para>
305     /// </summary>
306     /// <param name="first">
307     /// <para>The first.</para>
308     /// <para></para>
309     /// </param>
310     /// <param name="second">
311     /// <para>The second.</para>
312     /// <para></para>
313     /// </param>
314     /// <returns>
315     /// <para>The bool</para>
316     /// <para></para>
317     /// </returns>
318     [MethodImpl(MethodImplOptions.AggressiveInlining)]
319     protected override bool GreaterThan(ulong first, ulong second) => first > second;
320
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

```

```

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

```

```

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

```

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

```

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

```

```

11  /// Represents the int 64 unused links list methods.
12  /// </para>
13  /// <para></para>
14  /// </summary>
15  /// <seealso cref="UnusedLinksListMethods{TLinkAddress}"/>
16  public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<TLinkAddress>
17  {
18      private readonly RawLinkDataPart<ulong>* _links;
19      private readonly LinksHeader<ulong>* _header;
20
21      /// <summary>
22      /// <para>
23      /// Initializes a new <see cref="UInt64UnusedLinksListMethods"/> instance.
24      /// </para>
25      /// <para></para>
26      /// </summary>
27      /// <param name="links">
28      /// <para>A links.</para>
29      /// <para></para>
30      /// </param>
31      /// <param name="header">
32      /// <para>A header.</para>
33      /// <para></para>
34      /// </param>
35      [MethodImpl(MethodImplOptions.AggressiveInlining)]
36      public UInt64UnusedLinksListMethods(RawLinkDataPart<ulong>* links, LinksHeader<ulong>*
37      ↪ header)
38      : base((byte*)links, (byte*)header)
39      {
40      }
41
42      /// <summary>
43      /// <para>
44      /// Gets the link data part 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 data part of t link</para>
54      /// <para></para>
55      /// </returns>
56      [MethodImpl(MethodImplOptions.AggressiveInlining)]
57      protected override ref RawLinkDataPart<TLinkAddress>
58      ↪ 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> :
24         ↳ SizedAndThreadedAVLBalancedTreeMethods<TLinkAddress>, ILinksTreeMethods<TLinkAddress>
25     {
26         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
27             ↳ = UncheckedConverter<TLinkAddress, long>.Default;
28         private static readonly UncheckedConverter<TLinkAddress, int> _addressToInt32Converter =
29             ↳ UncheckedConverter<TLinkAddress, int>.Default;
30         private static readonly UncheckedConverter<bool, TLinkAddress> _boolToAddressConverter =
31             ↳ UncheckedConverter<bool, TLinkAddress>.Default;
32         private static readonly UncheckedConverter<TLinkAddress, bool> _addressToBoolConverter =
33             ↳ UncheckedConverter<TLinkAddress, bool>.Default;
34         private static readonly UncheckedConverter<int, TLinkAddress> _int32ToAddressConverter =
35             ↳ UncheckedConverter<int, TLinkAddress>.Default;
36
37         /// <summary>
38         /// <para>
39         /// The break.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         protected readonly TLinkAddress Break;
44
45         /// <summary>
46         /// <para>
47         /// The continue.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         protected readonly TLinkAddress Continue;
52
53         /// <summary>
54         /// <para>
55         /// The links.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         protected readonly byte* Links;
60
61         /// <summary>
62         /// <para>
63         /// The header.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         protected readonly byte* Header;
68
69         /// <summary>
70         /// <para>
71         /// Initializes a new <see cref="LinksAvlBalancedTreeMethodsBase"/> instance.
72         /// </para>
73         /// <para></para>
74         /// </summary>
75         /// <param name="constants">
76         /// <para>A constants.</para>
77         /// <para></para>
78         /// </param>
79         /// <param name="links">
80         /// <para>A links.</para>
81         /// <para></para>
82         /// </param>
83         /// <param name="header">
84         /// <para>A header.</para>
85         /// <para></para>
86         /// </param>
87         [MethodImpl(MethodImplOptions.AggressiveInlining)]
88         protected LinksAvlBalancedTreeMethodsBase(LinksConstants<TLinkAddress> constants, byte*
89             ↳ links, byte* header)
90         {
91             Links = links;
92             Header = header;
93             Break = constants.Break;
94             Continue = constants.Continue;
95         }
96     }
97 }

```



```

85     }
86
87     /// <summary>
88     /// <para>
89     /// Gets the tree root.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <returns>
94     /// <para>The link</para>
95     /// <para></para>
96     /// </returns>
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     protected abstract TLinkAddress GetTreeRoot();
99
100    /// <summary>
101    /// <para>
102    /// Gets the base part value using the specified link.
103    /// </para>
104    /// <para></para>
105    /// </summary>
106    /// <param name="link">
107    /// <para>The link.</para>
108    /// <para></para>
109    /// </param>
110    /// <returns>
111    /// <para>The link</para>
112    /// <para></para>
113    /// </returns>
114    [MethodImpl(MethodImplOptions.AggressiveInlining)]
115    protected abstract TLinkAddress GetBasePartValue(TLinkAddress link);
116
117    /// <summary>
118    /// <para>
119    /// 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 FirstIsToTheLeftOfSecond(TLinkAddress first, TLinkAddress second)
246 {
247     ref var firstLink = ref GetLinkReference(first);
248     ref var secondLink = ref GetLinkReference(second);
249     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
250         ↪ secondLink.Source, secondLink.Target);
251 }
252
253 /// <summary>
254 /// <para>
255 /// Determines whether this instance first is to the right of second.
256 /// </para>
257 /// <para></para>
258 /// </summary>
259 /// <param name="first">
260 /// <para>The first.</para>
261 /// <para></para>
262 /// </param>
263 /// <param name="second">
264 /// <para>The second.</para>
265 /// <para></para>
266 /// </param>
267 /// <returns>
268 /// <para>The bool</para>
269 /// <para></para>
270 /// </returns>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
273     ↪ second)
274 {
275     ref var firstLink = ref GetLinkReference(first);
276     ref var secondLink = ref GetLinkReference(second);
277     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
278         ↪ secondLink.Source, secondLink.Target);
279 }
280
281 /// <summary>
282 /// <para>
283 /// Gets the size value using the specified value.
284 /// </para>
285 /// <para></para>
286 /// </summary>
287 /// <param name="value">
288 /// <para>The value.</para>
289 /// <para></para>
290 /// </param>
291 /// <returns>
292 /// <para>The link</para>
293 /// <para></para>
294 /// </returns>
295 [MethodImpl(MethodImplOptions.AggressiveInlining)]
296 protected virtual TLinkAddress GetSizeValue(TLinkAddress value) =>
297     ↪ Bit<TLinkAddress>.PartialRead(value, 5, -5);
298
299 /// <summary>
300 /// <para>
301 /// Sets the size value using the specified stored value.
302 /// </para>
303 /// <para></para>
304 /// </summary>
305 /// <param name="storedValue">
306 /// <para>The stored value.</para>
307 /// <para></para>
308 /// </param>
309 /// <param name="size">
310 /// <para>The size.</para>
311 /// <para></para>
312 /// </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,
    ↪ 1));
332         //return !EqualityComparer.Equals(Bit<TLinkAddress>.PartialRead(value, 4, 1),
    ↪ default);
333     }
334 }
335
336 /// <summary>
337 /// <para>
338 /// Sets the left is child value using the specified stored value.
339 /// </para>
340 /// <para></para>
341 /// </summary>
342 /// <param name="storedValue">
343 /// <para>The stored value.</para>
344 /// <para></para>
345 /// </param>
346 /// <param name="value">
347 /// <para>The value.</para>
348 /// <para></para>
349 /// </param>
350 [MethodImpl(MethodImplOptions.AggressiveInlining)]
351 protected virtual void SetLeftIsChildValue(ref TLinkAddress storedValue, bool value)
352 {
353     unchecked
354     {
355         var previousValue = storedValue;
356         var modified = Bit<TLinkAddress>.PartialWrite(previousValue,
    ↪ _boolToAddressConverter.Convert(value), 4, 1);
357         storedValue = modified;
358     }
359 }
360
361 /// <summary>
362 /// <para>
363 /// Determines whether this instance get right is child value.
364 /// </para>
365 /// <para></para>
366 /// </summary>
367 /// <param name="value">
368 /// <para>The value.</para>
369 /// <para></para>
370 /// </param>
371 /// <returns>
372 /// <para>The bool</para>
373 /// <para></para>
374 /// </returns>
375 [MethodImpl(MethodImplOptions.AggressiveInlining)]
376 protected virtual bool GetRightIsChildValue(TLinkAddress value)
377 {
378     unchecked
379     {
380         return _addressToBoolConverter.Convert(Bit<TLinkAddress>.PartialRead(value, 3,
    ↪ 1));

```

```

381         //return !EqualityComparer.Equals(Bit<TLinkAddress>.PartialRead(value, 3, 1),
382         ↪ default);
383     }
384 }
385
386 /// <summary>
387 /// <para>
388 /// Sets the right is child value using the specified stored value.
389 /// </para>
390 /// <para></para>
391 /// </summary>
392 /// <param name="storedValue">
393 /// <para>The stored value.</para>
394 /// <para></para>
395 /// </param>
396 /// <param name="value">
397 /// <para>The value.</para>
398 /// <para></para>
399 /// </param>
400 [MethodImpl(MethodImplOptions.AggressiveInlining)]
401 protected virtual void SetRightIsChildValue(ref TLinkAddress storedValue, bool value)
402 {
403     unchecked
404     {
405         var previousValue = storedValue;
406         var modified = Bit<TLinkAddress>.PartialWrite(previousValue,
407             ↪ _boolToAddressConverter.Convert(value), 3, 1);
408         storedValue = modified;
409     }
410 }
411
412 /// <summary>
413 /// <para>
414 /// Determines whether this instance is child.
415 /// </para>
416 /// <para></para>
417 /// </summary>
418 /// <param name="parent">
419 /// <para>The parent.</para>
420 /// <para></para>
421 /// </param>
422 /// <param name="possibleChild">
423 /// <para>The possible child.</para>
424 /// <para></para>
425 /// </param>
426 /// <returns>
427 /// <para>The bool</para>
428 /// <para></para>
429 /// </returns>
430 [MethodImpl(MethodImplOptions.AggressiveInlining)]
431 protected bool IsChild(TLinkAddress parent, TLinkAddress possibleChild)
432 {
433     var parentSize = GetSize(parent);
434     var childSize = GetSizeOrZero(possibleChild);
435     return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
436 }
437
438 /// <summary>
439 /// <para>
440 /// Gets the balance value using the specified stored value.
441 /// </para>
442 /// <para></para>
443 /// </summary>
444 /// <param name="storedValue">
445 /// <para>The stored value.</para>
446 /// <para></para>
447 /// </param>
448 /// <returns>
449 /// <para>The sbyte</para>
450 /// <para></para>
451 /// </returns>
452 [MethodImpl(MethodImplOptions.AggressiveInlining)]
453 protected virtual sbyte GetBalanceValue(TLinkAddress storedValue)
454 {
455     unchecked
456     {

```

```

455     var value =
456         ↪ _addressToInt32Converter.Convert(Bit<TLinkAddress>.PartialRead(storedValue,
457         ↪ 0, 3));
458     value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
459     ↪ end of sbyte
460     return (sbyte)value;
461 }
462
463 /// <summary>
464 /// <para>
465 /// Sets the balance value using the specified stored value.
466 /// </para>
467 /// <para></para>
468 /// </summary>
469 /// <param name="storedValue">
470 /// <para>The stored value.</para>
471 /// <para></para>
472 /// </param>
473 /// <param name="value">
474 /// <para>The value.</para>
475 /// <para></para>
476 /// </param>
477 [MethodImpl(MethodImplOptions.AggressiveInlining)]
478 protected virtual void SetBalanceValue(ref TLinkAddress storedValue, sbyte value)
479 {
480     unchecked
481     {
482         var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
483         ↪ value & 3);
484         var modified = Bit<TLinkAddress>.PartialWrite(storedValue, packagedValue, 0, 3);
485         storedValue = modified;
486     }
487 }
488
489 /// <summary>
490 /// <para>
491 /// The zero.
492 /// </para>
493 /// <para></para>
494 /// </summary>
495 public TLinkAddress this[TLinkAddress index]
496 {
497     [MethodImpl(MethodImplOptions.AggressiveInlining)]
498     get
499     {
500         var root = GetTreeRoot();
501         if (GreaterOrEqualThan(index, GetSize(root)))
502         {
503             return Zero;
504         }
505         while (!EqualToZero(root))
506         {
507             var left = GetLeftOrDefault(root);
508             var leftSize = GetSizeOrZero(left);
509             if (LessThan(index, leftSize))
510             {
511                 root = left;
512                 continue;
513             }
514             if (AreEqual(index, leftSize))
515             {
516                 return root;
517             }
518             root = GetRightOrDefault(root);
519             index = Subtract(index, Increment(leftSize));
520         }
521         return Zero; // TODO: Impossible situation exception (only if tree structure
522         ↪ broken)
523     }
524 }
525
526 /// <summary>
527 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
528 ↪ (концом).
529 /// </summary>
530 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
531 /// <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)) //
538             ↪ node.Key < root.Key
539         {
540             root = GetLeftOrDefault(root);
541         }
542         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
543             ↪ node.Key > root.Key
544         {
545             root = GetRightOrDefault(root);
546         }
547         else // node.Key == root.Key
548         {
549             return root;
550         }
551     }
552     return Zero;
553 }
554
555 /// TODO: Return indices range instead of references count
556 /// <summary>
557 /// <para>
558 /// Counts the usages using the specified link.
559 /// </para>
560 /// <para></para>
561 /// </summary>
562 /// <param name="link">
563 /// <para>The link.</para>
564 /// <para></para>
565 /// </param>
566 /// <returns>
567 /// <para>The link</para>
568 /// <para></para>
569 /// </returns>
570 [MethodImpl(MethodImplOptions.AggressiveInlining)]
571 public TLinkAddress CountUsages(TLinkAddress link)
572 {
573     var root = GetTreeRoot();
574     var total = GetSize(root);
575     var totalRightIgnore = Zero;
576     while (!EqualToZero(root))
577     {
578         var @base = GetBasePartValue(root);
579         if (LessOrEqualThan(@base, link))
580         {
581             root = GetRightOrDefault(root);
582         }
583         else
584         {
585             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
586             root = GetLeftOrDefault(root);
587         }
588     }
589     root = GetTreeRoot();
590     var totalLeftIgnore = Zero;
591     while (!EqualToZero(root))
592     {
593         var @base = GetBasePartValue(root);
594         if (GreaterOrEqualThan(@base, link))
595         {
596             root = GetLeftOrDefault(root);
597         }
598         else
599         {
600             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
601             root = GetRightOrDefault(root);
602         }
603     }

```

```

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

```



```

681     /// </param>
682     [MethodImpl(MethodImplOptions.AggressiveInlining)]
683     protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
684     {
685         ref var link = ref GetLinkReference(node);
686         sb.Append(' ');
687         sb.Append(link.Source);
688         sb.Append('-');
689         sb.Append('>');
690         sb.Append(link.Target);
691     }
692 }
693 }

```

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

```

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

```

```

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

```

```

139     protected abstract bool FirstIsToTheRightOfSecond(TLinkAddress source, TLinkAddress
    ↪     target, TLinkAddress rootSource, TLinkAddress rootTarget);
140
141     /// <summary>
142     /// <para>
143     /// Determines whether this instance first is to the left of second.
144     /// </para>
145     /// <para></para>
146     /// </summary>
147     /// <param name="source">
148     /// <para>The source.</para>
149     /// <para></para>
150     /// </param>
151     /// <param name="target">
152     /// <para>The target.</para>
153     /// <para></para>
154     /// </param>
155     /// <param name="rootSource">
156     /// <para>The root source.</para>
157     /// <para></para>
158     /// </param>
159     /// <param name="rootTarget">
160     /// <para>The root target.</para>
161     /// <para></para>
162     /// </param>
163     /// <returns>
164     /// <para>The bool</para>
165     /// <para></para>
166     /// </returns>
167     [MethodImpl(MethodImplOptions.AggressiveInlining)]
168     protected abstract bool FirstIsToTheLeftOfSecond(TLinkAddress source, TLinkAddress
    ↪     target, TLinkAddress rootSource, TLinkAddress rootTarget);
169
170     /// <summary>
171     /// <para>
172     /// Gets the header reference.
173     /// </para>
174     /// <para></para>
175     /// </summary>
176     /// <returns>
177     /// <para>A ref links header of t link</para>
178     /// <para></para>
179     /// </returns>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     protected virtual ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
    ↪     AsRef<LinksHeader<TLinkAddress>>(Header);
182
183     /// <summary>
184     /// <para>
185     /// Gets the link reference using the specified link.
186     /// </para>
187     /// <para></para>
188     /// </summary>
189     /// <param name="link">
190     /// <para>The link.</para>
191     /// <para></para>
192     /// </param>
193     /// <returns>
194     /// <para>A ref raw link of t link</para>
195     /// <para></para>
196     /// </returns>
197     [MethodImpl(MethodImplOptions.AggressiveInlining)]
198     protected virtual ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
    ↪     AsRef<RawLink<TLinkAddress>>(Links + (RawLink<TLinkAddress>.SizeInBytes *
    ↪     _addressToInt64Converter.Convert(link)));
199
200     /// <summary>
201     /// <para>
202     /// Gets the link values using the specified link index.
203     /// </para>
204     /// <para></para>
205     /// </summary>
206     /// <param name="linkIndex">
207     /// <para>The link index.</para>
208     /// <para></para>
209     /// </param>
210     /// <returns>

```

```

211 /// <para>A list of t link</para>
212 /// <para></para>
213 /// </returns>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
216 {
217     ref var link = ref GetLinkReference(linkIndex);
218     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
219 }
220
221 /// <summary>
222 /// <para>
223 /// Determines whether this instance first is to the left of second.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="first">
228 /// <para>The first.</para>
229 /// <para></para>
230 /// </param>
231 /// <param name="second">
232 /// <para>The second.</para>
233 /// <para></para>
234 /// </param>
235 /// <returns>
236 /// <para>The bool</para>
237 /// <para></para>
238 /// </returns>
239 [MethodImpl(MethodImplOptions.AggressiveInlining)]
240 protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
241 {
242     ref var firstLink = ref GetLinkReference(first);
243     ref var secondLink = ref GetLinkReference(second);
244     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
245         ↪ secondLink.Source, secondLink.Target);
246 }
247
248 /// <summary>
249 /// <para>
250 /// Determines whether this instance first is to the right of second.
251 /// </para>
252 /// <para></para>
253 /// </summary>
254 /// <param name="first">
255 /// <para>The first.</para>
256 /// <para></para>
257 /// </param>
258 /// <param name="second">
259 /// <para>The second.</para>
260 /// <para></para>
261 /// </param>
262 /// <returns>
263 /// <para>The bool</para>
264 /// <para></para>
265 /// </returns>
266 [MethodImpl(MethodImplOptions.AggressiveInlining)]
267 protected override bool FirstIsToTheRightOfSecond(TLinkAddress first, TLinkAddress
268     ↪ second)
269 {
270     ref var firstLink = ref GetLinkReference(first);
271     ref var secondLink = ref GetLinkReference(second);
272     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
273         ↪ secondLink.Source, secondLink.Target);
274 }
275
276 /// <summary>
277 /// <para>
278 /// The zero.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 public TLinkAddress this[TLinkAddress index]
283 {
284     [MethodImpl(MethodImplOptions.AggressiveInlining)]
285     get
286     {
287         var root = GetTreeRoot();
288         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>
358 [MethodImpl(MethodImplOptions.AggressiveInlining)]
359 public TLinkAddress CountUsages(TLinkAddress link)
360 {
361     var root = GetTreeRoot();
362     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     else if (LessThan(linkBasePart, @base))
435     {
436         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
437         {
438             return @break;
439         }
440     }
441 }

```

```

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.

```

```

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

```



```

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

```

```

181 protected virtual ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
    ↳ AsRef<LinksHeader<TLinkAddress>>(Header);
182
183 /// <summary>
184 /// <para>
185 /// Gets the link reference using the specified link.
186 /// </para>
187 /// <para></para>
188 /// </summary>
189 /// <param name="link">
190 /// <para>The link.</para>
191 /// <para></para>
192 /// </param>
193 /// <returns>
194 /// <para>A ref raw link of t link</para>
195 /// <para></para>
196 /// </returns>
197 [MethodImpl(MethodImplOptions.AggressiveInlining)]
198 protected virtual ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
    ↳ AsRef<RawLink<TLinkAddress>>(Links + (RawLink<TLinkAddress>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));
199
200 /// <summary>
201 /// <para>
202 /// Gets the link values using the specified link index.
203 /// </para>
204 /// <para></para>
205 /// </summary>
206 /// <param name="linkIndex">
207 /// <para>The link index.</para>
208 /// <para></para>
209 /// </param>
210 /// <returns>
211 /// <para>A list of t link</para>
212 /// <para></para>
213 /// </returns>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 protected virtual IList<TLinkAddress>? GetLinkValues(TLinkAddress linkIndex)
216 {
217     ref var link = ref GetLinkReference(linkIndex);
218     return new Link<TLinkAddress>(linkIndex, link.Source, link.Target);
219 }
220
221 /// <summary>
222 /// <para>
223 /// Determines whether this instance first is to the left of second.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="first">
228 /// <para>The first.</para>
229 /// <para></para>
230 /// </param>
231 /// <param name="second">
232 /// <para>The second.</para>
233 /// <para></para>
234 /// </param>
235 /// <returns>
236 /// <para>The bool</para>
237 /// <para></para>
238 /// </returns>
239 [MethodImpl(MethodImplOptions.AggressiveInlining)]
240 protected override bool FirstIsToLeftOfSecond(TLinkAddress first, TLinkAddress second)
241 {
242     ref var firstLink = ref GetLinkReference(first);
243     ref var secondLink = ref GetLinkReference(second);
244     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↳ 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)) //
                ↪ node.Key < root.Key
325             {
326                 root = GetLeftOrDefault(root);
327             }

```

```

328         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
329             ↪ node.Key > root.Key
330         {
331             root = GetRightOrDefault(root);
332         }
333         else // node.Key == root.Key
334         {
335             return root;
336         }
337     }
338     return Zero;
339 }
340
341 // TODO: Return indices range instead of references count
342 /// <summary>
343 /// <para>
344 /// Counts the usages using the specified link.
345 /// </para>
346 /// <para></para>
347 /// </summary>
348 /// <param name="link">
349 /// <para>The link.</para>
350 /// <para></para>
351 /// </param>
352 /// <returns>
353 /// <para>The link</para>
354 /// <para></para>
355 /// </returns>
356 [MethodImpl(MethodImplOptions.AggressiveInlining)]
357 public TLinkAddress CountUsages(TLinkAddress link)
358 {
359     var root = GetTreeRoot();
360     var total = GetSize(root);
361     var totalRightIgnore = Zero;
362     while (!EqualToZero(root))
363     {
364         var @base = GetBasePartValue(root);
365         if (LessOrEqualThan(@base, link))
366         {
367             root = GetRightOrDefault(root);
368         }
369         else
370         {
371             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
372             root = GetLeftOrDefault(root);
373         }
374     }
375     root = GetTreeRoot();
376     var totalLeftIgnore = Zero;
377     while (!EqualToZero(root))
378     {
379         var @base = GetBasePartValue(root);
380         if (GreaterOrEqualThan(@base, link))
381         {
382             root = GetLeftOrDefault(root);
383         }
384         else
385         {
386             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
387             root = GetRightOrDefault(root);
388         }
389     }
390     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
391 }
392
393 /// <summary>
394 /// <para>
395 /// Eaches the usage using the specified base.
396 /// </para>
397 /// <para></para>
398 /// </summary>
399 /// <param name="@base">
400 /// <para>The base.</para>
401 /// <para></para>
402 /// </param>
403 /// <param name="handler">
404 /// <para>The handler.</para>
405 /// <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     else if (LessThan(linkBasePart, @base))
435     {
436         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
437         {
438             return @break;
439         }
440     }
441     else //if (linkBasePart == @base)
442     {
443         if (AreEqual(handler(GetLinkValues(link)), @break))
444         {
445             return @break;
446         }
447         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
448         {
449             return @break;
450         }
451         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
452         {
453             return @break;
454         }
455     }
456     return @continue;
457 }
458
459 /// <summary>
460 /// <para>
461 /// Prints the node value using the specified node.
462 /// </para>
463 /// <para></para>
464 /// </summary>
465 /// <param name="node">
466 /// <para>The node.</para>
467 /// <para></para>
468 /// </param>
469 /// <param name="sb">
470 /// <para>The sb.</para>
471 /// <para></para>
472 /// </param>
473 [MethodImpl(MethodImplOptions.AggressiveInlining)]
474 protected override void PrintNodeValue(TLinkAddress node, StringBuilder sb)
475 {
476     ref var link = ref GetLinkReference(node);
477     sb.Append(' ');
478     sb.Append(link.Source);
479     sb.Append('-');
480     sb.Append('>');
481     sb.Append(link.Target);
482 }

```

```
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        /// </param>
26        /// <param name="links">
27        /// <para>A links.</para>
28        /// </param>
29        /// <param name="header">
30        /// <para>A header.</para>
31        /// </param>
32        [MethodImpl(MethodImplOptions.AggressiveInlining)]
33        public LinksSourcesAvlBalancedTreeMethods(LinksConstants<TLinkAddress> constants, byte*
34            ↳ links, byte* header) : base(constants, links, header) { }
35
36        /// <summary>
37        /// <para>
38        /// Gets the left reference using the specified node.
39        /// </para>
40        /// <para></para>
41        /// </summary>
42        /// <param name="node">
43        /// <para>The node.</para>
44        /// </param>
45        /// <returns>
46        /// <para>The ref link</para>
47        /// </returns>
48        [MethodImpl(MethodImplOptions.AggressiveInlining)]
49        protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
50            ↳ GetLinkReference(node).LeftAsSource;
51
52        /// <summary>
53        /// <para>
54        /// Gets the right reference using the specified node.
55        /// </para>
56        /// <para></para>
57        /// </summary>
58        /// <param name="node">
59        /// <para>The node.</para>
60        /// </param>
61        /// <returns>
62        /// <para>The ref link</para>
63        /// </returns>
64        [MethodImpl(MethodImplOptions.AggressiveInlining)]
65        protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
66            ↳ GetLinkReference(node).RightAsSource;
67
68    }
69
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).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>

```

```

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

```



```

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

```

```

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

```

```

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/LinksSourcesRecursionlessSizeBalancedTreeMeth

```

using System.Runtime.CompilerServices;
1
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;

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```



```

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

```

```

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

```

```

1.86 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs
1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links targets avl balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="LinksAvlBalancedTreeMethodsBase{TLinkAddress}"/>
14     public unsafe class LinksTargetsAvlBalancedTreeMethods<TLinkAddress> :
15         ↳ LinksAvlBalancedTreeMethodsBase<TLinkAddress>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksTargetsAvlBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public LinksTargetsAvlBalancedTreeMethods(LinksConstants<TLinkAddress> constants, byte*
37             ↳ links, byte* header) : base(constants, links, header) { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="node">
46         /// <para>The node.</para>
47         /// <para></para>
48         /// </param>
49         /// <returns>
50         /// <para>The ref link</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref TLinkAddress GetLeftReference(TLinkAddress node) => ref
55             ↳ GetLinkReference(node).LeftAsTarget;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref link</para>
69         /// <para></para>
70         /// </returns>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
73             ↳ GetLinkReference(node).RightAsTarget;
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 link</para>
86         /// <para></para>
87         /// </returns>
88         [MethodImpl(MethodImplOptions.AggressiveInlining)]
89         protected override ref TLinkAddress GetLeftUsing(TLinkAddress node) => ref
90             ↳ GetLinkReference(node).LeftAsTarget;
91
92         /// <summary>
93         /// <para>
94         /// Gets the right using the specified node.
95         /// </para>
96         /// </summary>
97         /// <param name="node">
98         /// <para>The node.</para>
99         /// <para></para>
100        /// </param>
101        /// <returns>
102        /// <para>The ref link</para>
103        /// <para></para>
104        /// </returns>
105        [MethodImpl(MethodImplOptions.AggressiveInlining)]
106        protected override ref TLinkAddress GetRightUsing(TLinkAddress node) => ref
107            ↳ GetLinkReference(node).RightAsTarget;
108    }
109 }

```

```

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>

```

```

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

```

```

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

```

```

296     /// <para></para>
297     /// </param>
298     /// <returns>
299     /// <para>The link</para>
300     /// <para></para>
301     /// </returns>
302     [MethodImpl(MethodImplOptions.AggressiveInlining)]
303     protected override TLinkAddress GetBasePartValue(TLinkAddress link) =>
304         ↪ GetLinkReference(link).Target;
305
306     /// <summary>
307     /// <para>
308     /// Determines whether this instance first is to the left of second.
309     /// </para>
310     /// <para></para>
311     /// </summary>
312     /// <param name="firstSource">
313     /// <para>The first source.</para>
314     /// <para></para>
315     /// </param>
316     /// <param name="firstTarget">
317     /// <para>The first target.</para>
318     /// <para></para>
319     /// </param>
320     /// <param name="secondSource">
321     /// <para>The second source.</para>
322     /// <para></para>
323     /// </param>
324     /// <param name="secondTarget">
325     /// <para>The second target.</para>
326     /// <para></para>
327     /// </param>
328     /// <returns>
329     /// <para>The bool</para>
330     /// <para></para>
331     /// </returns>
332     [MethodImpl(MethodImplOptions.AggressiveInlining)]
333     protected override bool FirstIsToLeftOfSecond(TLinkAddress firstSource, TLinkAddress
334         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
335         ↪ LessThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
336         ↪ LessThan(firstSource, secondSource));
337
338     /// <summary>
339     /// <para>
340     /// Determines whether this instance first is to the right of second.
341     /// </para>
342     /// <para></para>
343     /// </summary>
344     /// <param name="firstSource">
345     /// <para>The first source.</para>
346     /// <para></para>
347     /// </param>
348     /// <param name="firstTarget">
349     /// <para>The first target.</para>
350     /// <para></para>
351     /// </param>
352     /// <param name="secondSource">
353     /// <para>The second source.</para>
354     /// <para></para>
355     /// </param>
356     /// <param name="secondTarget">
357     /// <para>The second target.</para>
358     /// <para></para>
359     /// </param>
360     /// <returns>
361     /// <para>The bool</para>
362     /// <para></para>
363     /// </returns>
364     [MethodImpl(MethodImplOptions.AggressiveInlining)]
365     protected override bool FirstIsToTheRightOfSecond(TLinkAddress firstSource, TLinkAddress
366         ↪ firstTarget, TLinkAddress secondSource, TLinkAddress secondTarget) =>
367         ↪ GreaterThan(firstTarget, secondTarget) || (AreEqual(firstTarget, secondTarget) &&
368         ↪ GreaterThan(firstSource, secondSource));
369
370     /// <summary>
371     /// <para>
372     /// 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> :
15         ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<TLinkAddress>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksTargetsRecursionlessSizeBalancedTreeMethods"/>
20         ↳ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// <para></para>
35         /// </param>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<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).LeftAsTarget;
57
58         /// <summary>
59         /// <para>

```



```

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

```

```

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

```

```

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

```

```

275         link.LeftAsTarget = Zero;
276         link.RightAsTarget = Zero;
277         link.SizeAsTarget = Zero;
278     }
279 }
280 }

```

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

```

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

```

```

68 [MethodImpl(MethodImplOptions.AggressiveInlining)]
69 protected override ref TLinkAddress GetRightReference(TLinkAddress node) => ref
    ↪ GetLinkReference(node).RightAsTarget;

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

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

104
105 /// <summary>
106 /// <para>
107 /// Sets the left using the specified node.
108 /// </para>
109 /// <para></para>
110 /// </summary>
111 /// <param name="node">
112 /// <para>The node.</para>
113 /// <para></para>
114 /// </param>
115 /// <param name="left">
116 /// <para>The left.</para>
117 /// <para></para>
118 /// </param>
119 [MethodImpl(MethodImplOptions.AggressiveInlining)]
120 protected override void SetLeft(TLinkAddress node, TLinkAddress left) =>
    ↪ GetLinkReference(node).LeftAsTarget = left;

121
122 /// <summary>
123 /// <para>
124 /// Sets the right using the specified node.
125 /// </para>
126 /// <para></para>
127 /// </summary>
128 /// <param name="node">
129 /// <para>The node.</para>
130 /// <para></para>
131 /// </param>
132 /// <param name="right">
133 /// <para>The right.</para>
134 /// <para></para>
135 /// </param>
136 [MethodImpl(MethodImplOptions.AggressiveInlining)]
137 protected override void SetRight(TLinkAddress node, TLinkAddress right) =>
    ↪ GetLinkReference(node).RightAsTarget = right;

138
139 /// <summary>

```

```

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

```

```

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

```

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

```

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

```

```

5 using static System.Runtime.CompilerServices.Unsafe;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Memory.United.Generic
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the united memory links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="UnitedMemoryLinksBase{TLinkAddress}"/>
18     public unsafe class UnitedMemoryLinks<TLinkAddress> : UnitedMemoryLinksBase<TLinkAddress>
19     {
20         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createSourceTreeMethods;
21         private readonly Func<ILinksTreeMethods<TLinkAddress>> _createTargetTreeMethods;
22         private byte* _header;
23         private byte* _links;
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="address">
32         /// <para>A address.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
37
38         /// <summary>
39         /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
40         /// → минимальным шагом расширения базы данных.
41         /// </summary>
42         /// <param name="address">Полный путь к файлу базы данных.</param>
43         /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
44         /// → байтах.</param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
47         {
48             FileMappedResizableDirectMemory(address, memoryReservationStep),
49             memoryReservationStep
50         }) { }
51
52         /// <summary>
53         /// <para>
54         /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <param name="memory">
59         /// <para>A memory.</para>
60         /// <para></para>
61         /// </param>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
64         {
65             DefaultLinksSizeStep
66         }) { }
67
68         /// <summary>
69         /// <para>
70         /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <param name="memory">
75         /// <para>A memory.</para>
76         /// <para></para>
77         /// </param>
78         /// <param name="memoryReservationStep">
79         /// <para>A memory reservation step.</para>
80         /// <para></para>
81         /// </param>
82         [MethodImpl(MethodImplOptions.AggressiveInlining)]
83         public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep) :
84         {
85             this(memory, memoryReservationStep, Default<LinksConstants<TLinkAddress>>.Instance,
86             IndexTreeType.Default) { }
87

```



```

75
76 /// <summary>
77 /// <para>
78 /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
79 /// </para>
80 /// <para></para>
81 /// </summary>
82 /// <param name="memory">
83 /// <para>A memory.</para>
84 /// <para></para>
85 /// </param>
86 /// <param name="memoryReservationStep">
87 /// <para>A memory reservation step.</para>
88 /// <para></para>
89 /// </param>
90 /// <param name="constants">
91 /// <para>A constants.</para>
92 /// <para></para>
93 /// </param>
94 /// <param name="indexTreeType">
95 /// <para>A index tree type.</para>
96 /// <para></para>
97 /// </param>
98 [MethodImpl(MethodImplOptions.AggressiveInlining)]
99 public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep,
   ↳ LinksConstants<TLinkAddress> constants, IndexTreeType indexTreeType) : base(memory,
   ↳ memoryReservationStep, constants)
100 {
101     if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
102     {
103         _createSourceTreeMethods = () => new
104             ↳ LinksSourcesAvlBalancedTreeMethods<TLinkAddress>(Constants, _links, _header);
105         _createTargetTreeMethods = () => new
106             ↳ LinksTargetsAvlBalancedTreeMethods<TLinkAddress>(Constants, _links, _header);
107     }
108     else
109     {
110         _createSourceTreeMethods = () => new
111             ↳ LinksSourcesSizeBalancedTreeMethods<TLinkAddress>(Constants, _links,
112             ↳ _header);
113         _createTargetTreeMethods = () => new
114             ↳ LinksTargetsSizeBalancedTreeMethods<TLinkAddress>(Constants, _links,
115             ↳ _header);
116     }
117     Init(memory, memoryReservationStep);
118 }
119
120 /// <summary>
121 /// <para>
122 /// Sets the pointers using the specified memory.
123 /// </para>
124 /// <para></para>
125 /// </summary>
126 /// <param name="memory">
127 /// <para>The memory.</para>
128 /// <para></para>
129 /// </param>
130 [MethodImpl(MethodImplOptions.AggressiveInlining)]
131 protected override void SetPointers(IResizableDirectMemory memory)
132 {
133     _links = (byte*)memory.Pointer;
134     _header = _links;
135     SourcesTreeMethods = _createSourceTreeMethods();
136     TargetsTreeMethods = _createTargetTreeMethods();
137     UnusedLinksListMethods = new UnusedLinksListMethods<TLinkAddress>(_links, _header);
138 }
139
140 /// <summary>
141 /// <para>
142 /// Resets the pointers.
143 /// </para>
144 /// <para></para>
145 /// </summary>
146 [MethodImpl(MethodImplOptions.AggressiveInlining)]
147 protected override void ResetPointers()
148 {
149     base.ResetPointers();
150 }

```

```

144         _links = null;
145         _header = null;
146     }
147
148     /// <summary>
149     /// <para>
150     /// Gets the header reference.
151     /// </para>
152     /// <para></para>
153     /// </summary>
154     /// <returns>
155     /// <para>A ref links header of t link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
160         ↪ AsRef<LinksHeader<TLinkAddress>>(_header);
161
162     /// <summary>
163     /// <para>
164     /// Gets the link reference using the specified link index.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="linkIndex">
169     /// <para>The link index.</para>
170     /// <para></para>
171     /// </param>
172     /// <returns>
173     /// <para>A ref raw link of t link</para>
174     /// <para></para>
175     /// </returns>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress linkIndex) =>
178         ↪ ref AsRef<RawLink<TLinkAddress>>(_links + (LinkSizeInBytes *
179         ↪ ConvertToInt64(linkIndex)));
180 }

```

## 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,
25         ↪ ILinks<TLinkAddress> where TLinkAddress : struct
26     {
27         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
28             ↪ EqualityComparer<TLinkAddress>.Default;
29         private static readonly Comparer<TLinkAddress> _comparer =
30             ↪ Comparer<TLinkAddress>.Default;
31         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
32             ↪ = UncheckedConverter<TLinkAddress, long>.Default;
33         private static readonly UncheckedConverter<long, TLinkAddress> _int64ToAddressConverter
34             ↪ = UncheckedConverter<long, TLinkAddress>.Default;
35         private static readonly TLinkAddress _zero = default;
36         private static readonly TLinkAddress _one = Arithmetic.Increment(_zero);
37
38         /// <summary>Возвращает размер одной связи в байтах.</summary>
39         /// <remarks>

```

```

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

```

```

111     /// <para></para>
112     /// </summary>
113     public virtual LinksConstants<TLinkAddress> Constants
114     {
115         [MethodImpl(MethodImplOptions.AggressiveInlining)]
116         get;
117     }
118
119     /// <summary>
120     /// <para>
121     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
122     /// </para>
123     /// <para></para>
124     /// </summary>
125     /// <param name="memory">
126     /// <para>A memory.</para>
127     /// <para></para>
128     /// </param>
129     /// <param name="memoryReservationStep">
130     /// <para>A memory reservation step.</para>
131     /// <para></para>
132     /// </param>
133     /// <param name="constants">
134     /// <para>A constants.</para>
135     /// <para></para>
136     /// </param>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
139     → memoryReservationStep, LinksConstants<TLinkAddress> constants)
140     {
141         _memory = memory;
142         _memoryReservationStep = memoryReservationStep;
143         Constants = constants;
144     }
145
146     /// <summary>
147     /// <para>
148     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="memory">
153     /// <para>A memory.</para>
154     /// <para></para>
155     /// </param>
156     /// <param name="memoryReservationStep">
157     /// <para>A memory reservation step.</para>
158     /// <para></para>
159     /// </param>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
162     → memoryReservationStep) : this(memory, memoryReservationStep,
163     → Default<LinksConstants<TLinkAddress>>.Instance) { }
164
165     /// <summary>
166     /// <para>
167     /// Inits the memory.
168     /// </para>
169     /// <para></para>
170     /// </summary>
171     /// <param name="memory">
172     /// <para>The memory.</para>
173     /// <para></para>
174     /// </param>
175     /// <param name="memoryReservationStep">
176     /// <para>The memory reservation step.</para>
177     /// <para></para>
178     /// </param>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
181     {
182         if (memory.ReservedCapacity < memoryReservationStep)
183         {
184             memory.ReservedCapacity = memoryReservationStep;
185         }
186         SetPointers(memory);
187         ref var header = ref GetHeaderReference();
188         // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks

```

```

186     memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
187         ↳ LinkHeaderSizeInBytes;
188     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
189     header.ReservedLinks = ConvertToAddress((memory.ReservedCapacity -
190         ↳ LinkHeaderSizeInBytes) / LinkSizeInBytes);
191 }
192
193 /// <summary>
194 /// <para>
195 /// Counts the substitution.
196 /// </para>
197 /// <para></para>
198 /// </summary>
199 /// <param name="restriction">
200 /// <para>The substitution.</para>
201 /// </param>
202 /// <exception cref="NotSupportedException">
203 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
204 /// </exception>
205 /// <returns>
206 /// <para>The link</para>
207 /// <para></para>
208 /// </returns>
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 public virtual TLinkAddress Count(ICollection<TLinkAddress>? restriction)
211 {
212     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
213     if (restriction.Count == 0)
214     {
215         return Total;
216     }
217     var constants = Constants;
218     var any = constants.Any;
219     var index = restriction[constants.IndexPart];
220     if (restriction.Count == 1)
221     {
222         if (AreEqual(index, any))
223         {
224             return Total;
225         }
226         return Exists(index) ? GetOne() : GetZero();
227     }
228     if (restriction.Count == 2)
229     {
230         var value = restriction[1];
231         if (AreEqual(index, any))
232         {
233             if (AreEqual(value, any))
234             {
235                 return Total; // Any - как отсутствие ограничения
236             }
237             return Add(SourcesTreeMethods.CountUsages(value),
238                 ↳ TargetsTreeMethods.CountUsages(value));
239         }
240         else
241         {
242             if (!Exists(index))
243             {
244                 return GetZero();
245             }
246             if (AreEqual(value, any))
247             {
248                 return GetOne();
249             }
250             ref var storedLinkValue = ref GetLinkReference(index);
251             if (AreEqual(storedLinkValue.Source, value) ||
252                 ↳ AreEqual(storedLinkValue.Target, value))
253             {
254                 return GetOne();
255             }
256             return GetZero();
257         }
258     }
259     if (restriction.Count == 3)
260     {
261         var source = restriction[constants.SourcePart];

```

```

260     var target = restriction[constants.TargetPart];
261     if (AreEqual(index, any))
262     {
263         if (AreEqual(source, any) && AreEqual(target, any))
264         {
265             return Total;
266         }
267         else if (AreEqual(source, any))
268         {
269             return TargetsTreeMethods.CountUsages(target);
270         }
271         else if (AreEqual(target, any))
272         {
273             return SourcesTreeMethods.CountUsages(source);
274         }
275         else //if(source != Any && target != Any)
276         {
277             // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
278             var link = SourcesTreeMethods.Search(source, target);
279             return AreEqual(link, constants.Null) ? GetZero() : GetOne();
280         }
281     }
282     else
283     {
284         if (!Exists(index))
285         {
286             return GetZero();
287         }
288         if (AreEqual(source, any) && AreEqual(target, any))
289         {
290             return GetOne();
291         }
292         ref var storedLinkValue = ref GetLinkReference(index);
293         if (!AreEqual(source, any) && !AreEqual(target, any))
294         {
295             if (AreEqual(storedLinkValue.Source, source) &&
296                 ⇨ AreEqual(storedLinkValue.Target, target))
297             {
298                 return GetOne();
299             }
300             return GetZero();
301         }
302         var value = default(TLinkAddress);
303         if (AreEqual(source, any))
304         {
305             value = target;
306         }
307         if (AreEqual(target, any))
308         {
309             value = source;
310         }
311         if (AreEqual(storedLinkValue.Source, value) ||
312             ⇨ AreEqual(storedLinkValue.Target, value))
313         {
314             return GetOne();
315         }
316         return GetZero();
317     }
318 }
319
320 throw new NotSupportedException("Другие размеры и способы ограничений не
321 ⇨ поддерживаются.");
322
323 /// <summary>
324 /// <para>
325 /// Eaches the handler.
326 /// </para>
327 /// <para></para>
328 /// </summary>
329 /// <param name="handler">
330 /// <para>The handler.</para>
331 /// <para></para>
332 /// </param>
333 /// <param name="restriction">
334 /// <para>The substitution.</para>
335 /// <para></para>
336 /// </param>
337 /// <exception cref="NotSupportedException">

```

```

335 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
336 /// <para></para>
337 /// </exception>
338 /// <returns>
339 /// <para>The link</para>
340 /// <para></para>
341 /// </returns>
342 [MethodImpl(MethodImplOptions.AggressiveInlining)]
343 public virtual TLinkAddress Each(IList<TLinkAddress>? restriction,
    ↳ ReadHandler<TLinkAddress>? handler)
344 {
345     var constants = Constants;
346     var @break = constants.Break;
347     if (restriction.Count == 0)
348     {
349         for (var link = GetOne(); LessOrEqualThan(link,
    ↳ GetHeaderReference().AllocatedLinks); link = Increment(link))
350         {
351             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
352             {
353                 return @break;
354             }
355         }
356         return @break;
357     }
358     var @continue = constants.Continue;
359     var any = constants.Any;
360     var index = restriction[constants.IndexPart];
361     if (restriction.Count == 1)
362     {
363         if (AreEqual(index, any))
364         {
365             return Each(Array.Empty<TLinkAddress>(), handler);
366         }
367         if (!Exists(index))
368         {
369             return @continue;
370         }
371         return handler(GetLinkStruct(index));
372     }
373     if (restriction.Count == 2)
374     {
375         var value = restriction[1];
376         if (AreEqual(index, any))
377         {
378             if (AreEqual(value, any))
379             {
380                 return Each(Array.Empty<TLinkAddress>(), handler);
381             }
382             if (AreEqual(Each(new Link<TLinkAddress>(index, value, any), handler),
    ↳ @break))
383             {
384                 return @break;
385             }
386             return Each(new Link<TLinkAddress>(index, any, value), handler);
387         }
388         else
389         {
390             if (!Exists(index))
391             {
392                 return @continue;
393             }
394             if (AreEqual(value, any))
395             {
396                 return handler(GetLinkStruct(index));
397             }
398             ref var storedLinkValue = ref GetLinkReference(index);
399             if (AreEqual(storedLinkValue.Source, value) ||
400                 AreEqual(storedLinkValue.Target, value))
401             {
402                 return handler(GetLinkStruct(index));
403             }
404             return @continue;
405         }
406     }
407     if (restriction.Count == 3)
408     {
409         var source = restriction[constants.SourcePart];

```

```

410     var target = restriction[constants.TargetPart];
411     if (AreEqual(index, any))
412     {
413         if (AreEqual(source, any) && AreEqual(target, any))
414         {
415             return Each(Array.Empty<TLinkAddress>(), handler);
416         }
417         else if (AreEqual(source, any))
418         {
419             return TargetsTreeMethods.EachUsage(target, handler);
420         }
421         else if (AreEqual(target, any))
422         {
423             return SourcesTreeMethods.EachUsage(source, handler);
424         }
425         else //if(source != Any && target != Any)
426         {
427             var link = SourcesTreeMethods.Search(source, target);
428             return AreEqual(link, constants.Null) ? @continue :
429                 ↪ handler(GetLinkStruct(link));
430         }
431     }
432     else
433     {
434         if (!Exists(index))
435         {
436             return @continue;
437         }
438         if (AreEqual(source, any) && AreEqual(target, any))
439         {
440             return handler(GetLinkStruct(index));
441         }
442         ref var storedLinkValue = ref GetLinkReference(index);
443         if (!AreEqual(source, any) && !AreEqual(target, any))
444         {
445             if (AreEqual(storedLinkValue.Source, source) &&
446                 AreEqual(storedLinkValue.Target, target))
447             {
448                 return handler(GetLinkStruct(index));
449             }
450             return @continue;
451         }
452         var value = default(TLinkAddress);
453         if (AreEqual(source, any))
454         {
455             value = target;
456         }
457         if (AreEqual(target, any))
458         {
459             value = source;
460         }
461         if (AreEqual(storedLinkValue.Source, value) ||
462             AreEqual(storedLinkValue.Target, value))
463         {
464             return handler(GetLinkStruct(index));
465         }
466         return @continue;
467     }
468     throw new NotSupportedException("Другие размеры и способы ограничений не
469     ↪ поддерживаются.");
470 }
471
472 /// <remarks>
473 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
474 /// ↪ в другом месте (но не в менеджере памяти, а в логике Links)
475 /// </remarks>
476 [MethodImpl(MethodImplOptions.AggressiveInlining)]
477 public virtual TLinkAddress Update(IList<TLinkAddress>? restriction,
478     ↪ IList<TLinkAddress>? substitution, WriteHandler<TLinkAddress>? handler)
479 {
480     var constants = Constants;
481     var @null = constants.Null;
482     var linkIndex = restriction[constants.IndexPart];
483     var before = GetLinkStruct(linkIndex);
484     ref var link = ref GetLinkReference(linkIndex);
485     ref var header = ref GetHeaderReference();
486     ref var firstAsSource = ref header.RootAsSource;

```



```

484     ref var firstAsTarget = ref header.RootAsTarget;
485     // Будет корректно работать только в том случае, если пространство выделенной связи
486     ↪ предварительно заполнено нулями
487     if (!AreEqual(link.Source, @null))
488     {
489         SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
490     }
491     if (!AreEqual(link.Target, @null))
492     {
493         TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
494     }
495     link.Source = substitution[constants.SourcePart];
496     link.Target = substitution[constants.TargetPart];
497     if (!AreEqual(link.Source, @null))
498     {
499         SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
500     }
501     if (!AreEqual(link.Target, @null))
502     {
503         TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
504     }
505     return handler?.Invoke(before, GetLinkStruct(linkIndex)) ?? Constants.Continue;
506 }
507
508 /// <remarks>
509 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
510 ↪ пространство
511 /// </remarks>
512 [MethodImpl(MethodImplOptions.AggressiveInlining)]
513 public virtual TLinkAddress Create(IList<TLinkAddress>? substitution,
514     ↪ WriteHandler<TLinkAddress>? handler)
515 {
516     ref var header = ref GetHeaderReference();
517     var freeLink = header.FirstFreeLink;
518     if (!AreEqual(freeLink, Constants.Null))
519     {
520         UnusedLinksListMethods.Detach(freeLink);
521     }
522     else
523     {
524         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
525         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
526         {
527             throw new
528                 ↪ LinksLimitReachedException<TLinkAddress>(maximumPossibleInnerReference);
529         }
530         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
531         {
532             _memory.ReservedCapacity += _memory.ReservationStep;
533             SetPointers(_memory);
534             header = ref GetHeaderReference();
535             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
536                 ↪ LinkSizeInBytes);
537         }
538         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
539         _memory.UsedCapacity += LinkSizeInBytes;
540     }
541     return handler?.Invoke(null, new Link<TLinkAddress>(freeLink, Constants.Null,
542         ↪ Constants.Null)) ?? Constants.Continue;
543 }
544
545 /// <summary>
546 /// <para>
547 /// Deletes the substitution.
548 /// </para>
549 /// <para></para>
550 /// </summary>
551 /// <param name="restriction">
552 /// <para>The substitution.</para>
553 /// <para></para>
554 /// </param>
555 [MethodImpl(MethodImplOptions.AggressiveInlining)]
556 public virtual TLinkAddress Delete(IList<TLinkAddress>? restriction,
557     ↪ WriteHandler<TLinkAddress>? handler)
558 {
559     ref var header = ref GetHeaderReference();
560     var link = restriction[Constants.IndexPart];
561     var before = GetLinkStruct(link);

```

```

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

```

```

630    /// <para></para>
631    /// </returns>
632    [MethodImpl(MethodImplOptions.AggressiveInlining)]
633    protected abstract ref LinksHeader<TLinkAddress> GetHeaderReference();
634
635    /// <summary>
636    /// <para>
637    /// Gets the link reference using the specified link index.
638    /// </para>
639    /// <para></para>
640    /// </summary>
641    /// <param name="linkIndex">
642    /// <para>The link index.</para>
643    /// <para></para>
644    /// </param>
645    /// <returns>
646    /// <para>A ref raw link of t link</para>
647    /// <para></para>
648    /// </returns>
649    [MethodImpl(MethodImplOptions.AggressiveInlining)]
650    protected abstract ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress linkIndex);
651
652    /// <summary>
653    /// <para>
654    /// Determines whether this instance exists.
655    /// </para>
656    /// <para></para>
657    /// </summary>
658    /// <param name="link">
659    /// <para>The link.</para>
660    /// <para></para>
661    /// </param>
662    /// <returns>
663    /// <para>The bool</para>
664    /// <para></para>
665    /// </returns>
666    [MethodImpl(MethodImplOptions.AggressiveInlining)]
667    protected virtual bool Exists(TLinkAddress link)
668        => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
669            && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
670            && !IsUnusedLink(link);
671
672    /// <summary>
673    /// <para>
674    /// Determines whether this instance is unused link.
675    /// </para>
676    /// <para></para>
677    /// </summary>
678    /// <param name="linkIndex">
679    /// <para>The link index.</para>
680    /// <para></para>
681    /// </param>
682    /// <returns>
683    /// <para>The bool</para>
684    /// <para></para>
685    /// </returns>
686    [MethodImpl(MethodImplOptions.AggressiveInlining)]
687    protected virtual bool IsUnusedLink(TLinkAddress linkIndex)
688    {
689        if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
690            ↪ is not needed
691        {
692            ref var link = ref GetLinkReference(linkIndex);
693            return AreEqual(link.SizeAsSource, default) && !AreEqual(link.Source, default);
694        }
695        else
696        {
697            return true;
698        }
699    }
700
701    /// <summary>
702    /// <para>
703    /// Gets the one.
704    /// </para>
705    /// <para></para>
706    /// </summary>
707    /// </returns>

```

```

707     /// <para>The link</para>
708     /// <para></para>
709     /// </returns>
710     [MethodImpl(MethodImplOptions.AggressiveInlining)]
711     protected virtual TLinkAddress GetOne() => _one;
712
713     /// <summary>
714     /// <para>
715     /// Gets the zero.
716     /// </para>
717     /// <para></para>
718     /// </summary>
719     /// <returns>
720     /// <para>The link</para>
721     /// <para></para>
722     /// </returns>
723     [MethodImpl(MethodImplOptions.AggressiveInlining)]
724     protected virtual TLinkAddress GetZero() => default;
725
726     /// <summary>
727     /// <para>
728     /// Determines whether this instance are equal.
729     /// </para>
730     /// <para></para>
731     /// </summary>
732     /// <param name="first">
733     /// <para>The first.</para>
734     /// <para></para>
735     /// </param>
736     /// <param name="second">
737     /// <para>The second.</para>
738     /// <para></para>
739     /// </param>
740     /// <returns>
741     /// <para>The bool</para>
742     /// <para></para>
743     /// </returns>
744     [MethodImpl(MethodImplOptions.AggressiveInlining)]
745     protected virtual bool AreEqual(TLinkAddress first, TLinkAddress second) =>
746         ↪ _equalityComparer.Equals(first, second);
747
748     /// <summary>
749     /// <para>
750     /// Determines whether this instance less than.
751     /// </para>
752     /// <para></para>
753     /// </summary>
754     /// <param name="first">
755     /// <para>The first.</para>
756     /// <para></para>
757     /// </param>
758     /// <param name="second">
759     /// <para>The second.</para>
760     /// <para></para>
761     /// </param>
762     /// <returns>
763     /// <para>The bool</para>
764     /// <para></para>
765     /// </returns>
766     [MethodImpl(MethodImplOptions.AggressiveInlining)]
767     protected virtual bool LessThan(TLinkAddress first, TLinkAddress second) =>
768         ↪ _comparer.Compare(first, second) < 0;
769
770     /// <summary>
771     /// <para>
772     /// Determines whether this instance less or equal than.
773     /// </para>
774     /// <para></para>
775     /// </summary>
776     /// <param name="first">
777     /// <para>The first.</para>
778     /// <para></para>
779     /// </param>
780     /// <param name="second">
781     /// <para>The second.</para>
782     /// <para></para>
783     /// </param>
784     /// <returns>

```

```

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

```

```

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

```

```

931     /// <para>The link.</para>
932     /// <para></para>
933     /// </param>
934     /// <returns>
935     /// <para>The link</para>
936     /// <para></para>
937     /// </returns>
938     [MethodImpl(MethodImplOptions.AggressiveInlining)]
939     protected virtual TLinkAddress Decrement(TLinkAddress link) =>
        ↪ Arithmetic<TLinkAddress>.Decrement(link);
940
941     #region Disposable
942
943     /// <summary>
944     /// <para>
945     /// Gets the allow multiple dispose calls value.
946     /// </para>
947     /// <para></para>
948     /// </summary>
949     protected override bool AllowMultipleDisposeCalls
950     {
951         [MethodImpl(MethodImplOptions.AggressiveInlining)]
952         get => true;
953     }
954
955     /// <summary>
956     /// <para>
957     /// Disposes the manual.
958     /// </para>
959     /// <para></para>
960     /// </summary>
961     /// <param name="manual">
962     /// <para>The manual.</para>
963     /// <para></para>
964     /// </param>
965     /// <param name="wasDisposed">
966     /// <para>The was disposed.</para>
967     /// <para></para>
968     /// </param>
969     [MethodImpl(MethodImplOptions.AggressiveInlining)]
970     protected override void Dispose(bool manual, bool wasDisposed)
971     {
972         if (!wasDisposed)
973         {
974             ResetPointers();
975             _memory.DisposeIfPossible();
976         }
977     }
978
979     #endregion
980 }
981 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Methods.Lists;
3  using Platform.Converters;
4  using static System.Runtime.CompilerServices.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.United.Generic
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unused links list methods.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="AbsoluteCircularDoublyLinkedListMethods{TLinkAddress}"/>
17     /// <seealso cref="ILinksListMethods{TLinkAddress}"/>
18     public unsafe class UnusedLinksListMethods<TLinkAddress> :
        ↪ AbsoluteCircularDoublyLinkedListMethods<TLinkAddress>, ILinksListMethods<TLinkAddress>
19     {
20         private static readonly UncheckedConverter<TLinkAddress, long> _addressToInt64Converter
            ↪ = UncheckedConverter<TLinkAddress, long>.Default;
21         private readonly byte* _links;
22         private readonly byte* _header;
23

```

```

24     /// <summary>
25     /// <para>
26     /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
27     /// </para>
28     /// <para></para>
29     /// </summary>
30     /// <param name="links">
31     /// <para>A links.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="header">
35     /// <para>A header.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public UnusedLinksListMethods(byte* links, byte* header)
40     {
41         _links = links;
42         _header = header;
43     }
44
45     /// <summary>
46     /// <para>
47     /// Gets the header reference.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     /// <returns>
52     /// <para>A ref links header of t link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected virtual ref LinksHeader<TLinkAddress> GetHeaderReference() => ref
57     ↪ AsRef<LinksHeader<TLinkAddress>>(_header);
58
59     /// <summary>
60     /// <para>
61     /// Gets the link reference using the specified link.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="link">
66     /// <para>The link.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>A ref raw link of t link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected virtual ref RawLink<TLinkAddress> GetLinkReference(TLinkAddress link) => ref
75     ↪ AsRef<RawLink<TLinkAddress>>(_links + (RawLink<TLinkAddress>.SizeInBytes *
76     ↪ _addressToInt64Converter.Convert(link)));
77
78     /// <summary>
79     /// <para>
80     /// Gets the first.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <returns>
85     /// <para>The link</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override TLinkAddress GetFirst() => GetHeaderReference().FirstFreeLink;
90
91     /// <summary>
92     /// <para>
93     /// Gets the last.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <returns>
98     /// <para>The link</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

99     protected override TLinkAddress GetLast() => GetHeaderReference().LastFreeLink;
100
101     /// <summary>
102     /// <para>
103     /// Gets the previous using the specified element.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="element">
108     /// <para>The element.</para>
109     /// <para></para>
110     /// </param>
111     /// <returns>
112     /// <para>The link</para>
113     /// <para></para>
114     /// </returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     protected override TLinkAddress GetPrevious(TLinkAddress element) =>
117         ↪ GetLinkReference(element).Source;
118
119     /// <summary>
120     /// <para>
121     /// Gets the next using the specified element.
122     /// </para>
123     /// <para></para>
124     /// </summary>
125     /// <param name="element">
126     /// <para>The element.</para>
127     /// <para></para>
128     /// </param>
129     /// <returns>
130     /// <para>The link</para>
131     /// <para></para>
132     /// </returns>
133     [MethodImpl(MethodImplOptions.AggressiveInlining)]
134     protected override TLinkAddress GetNext(TLinkAddress element) =>
135         ↪ GetLinkReference(element).Target;
136
137     /// <summary>
138     /// <para>
139     /// Gets the size.
140     /// </para>
141     /// <para></para>
142     /// </summary>
143     /// <returns>
144     /// <para>The link</para>
145     /// <para></para>
146     /// </returns>
147     [MethodImpl(MethodImplOptions.AggressiveInlining)]
148     protected override TLinkAddress GetSize() => GetHeaderReference().FreeLinks;
149
150     /// <summary>
151     /// <para>
152     /// Sets the first using the specified element.
153     /// </para>
154     /// <para></para>
155     /// </summary>
156     /// <param name="element">
157     /// <para>The element.</para>
158     /// <para></para>
159     /// </param>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     protected override void SetFirst(TLinkAddress element) =>
162         ↪ GetHeaderReference().FirstFreeLink = element;
163
164     /// <summary>
165     /// <para>
166     /// Sets the last using the specified element.
167     /// </para>
168     /// <para></para>
169     /// </summary>
170     /// <param name="element">
171     /// <para>The element.</para>
172     /// <para></para>
173     /// </param>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     protected override void SetLast(TLinkAddress element) =>
176         ↪ GetHeaderReference().LastFreeLink = element;

```

```

173     /// <summary>
174     /// <para>
175     /// Sets the previous using the specified element.
176     /// </para>
177     /// <para></para>
178     /// </summary>
179     /// <param name="element">
180     /// <para>The element.</para>
181     /// <para></para>
182     /// </param>
183     /// <param name="previous">
184     /// <para>The previous.</para>
185     /// <para></para>
186     /// </param>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     protected override void SetPrevious(TLinkAddress element, TLinkAddress previous) =>
189     ↪ GetLinkReference(element).Source = previous;
190
191     /// <summary>
192     /// <para>
193     /// Sets the next using the specified element.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="element">
198     /// <para>The element.</para>
199     /// <para></para>
200     /// </param>
201     /// <param name="next">
202     /// <para>The next.</para>
203     /// <para></para>
204     /// </param>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override void SetNext(TLinkAddress element, TLinkAddress next) =>
207     ↪ GetLinkReference(element).Target = next;
208
209     /// <summary>
210     /// <para>
211     /// Sets the size using the specified size.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="size">
216     /// <para>The size.</para>
217     /// <para></para>
218     /// </param>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override void SetSize(TLinkAddress size) => GetHeaderReference().FreeLinks =
221     ↪ size;
222 }

```

## 1.92 ./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 =
19         ↪ EqualityComparer<TLinkAddress>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </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     /// <summary>
102     /// <para>
103     /// Determines whether this instance equals.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="other">
108     /// <para>The other.</para>
109     /// <para></para>
110     /// </param>
111     /// <returns>
112     /// <para>The bool</para>
113     /// <para></para>
114     /// </returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     public bool Equals(RawLink<TLinkAddress> other)
117     => _equalityComparer.Equals(Source, other.Source)
118         && _equalityComparer.Equals(Target, other.Target)
119         && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
120         && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
121         && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
122         && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
123         && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
124         && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);
125
126     /// <summary>
127     /// <para>
128     /// Gets the hash code.
129     /// </para>
130     /// <para></para>
131     /// </summary>
132     /// <returns>
133     /// <para>The int</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     public override int GetHashCode() => (Source, Target, LeftAsSource, RightAsSource,
138     ↪ SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
139
140     [MethodImpl(MethodImplOptions.AggressiveInlining)]
141     public static bool operator ==(RawLink<TLinkAddress> left, RawLink<TLinkAddress> right)
142     ↪ => left.Equals(right);
143
144     [MethodImpl(MethodImplOptions.AggressiveInlining)]
145     public static bool operator !=(RawLink<TLinkAddress> left, RawLink<TLinkAddress> right)
146     ↪ => !(left == right);
147 }
148 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 32 links recursionless size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{uint}"/>
15     public unsafe abstract class UInt32LinksRecursionlessSizeBalancedTreeMethodsBase :
16     ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<uint>
17     {
18         /// <summary>
19         /// <para>
20         /// The links.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         protected new readonly RawLink<uint>* Links;
25
26         /// <summary>
27         /// <para>
28         /// The header.
29         /// </para>
30         /// </summary>
31         protected new readonly RawLink<uint>* Header;
32     }
33 }

```

```

27     /// </para>
28     /// <para></para>
29     /// </summary>
30     protected new readonly LinksHeader<uint>* Header;
31
32     /// <summary>
33     /// <para>
34     /// Initializes a new <see cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
35     ↪ instance.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     /// <param name="constants">
40     /// <para>A constants.</para>
41     /// <para></para>
42     /// </param>
43     /// <param name="links">
44     /// <para>A links.</para>
45     /// <para></para>
46     /// </param>
47     /// <param name="header">
48     /// <para>A header.</para>
49     /// <para></para>
50     /// </param>
51     protected UInt32LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<uint>
52     ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header)
53     : base(constants, (byte*)links, (byte*)header)
54     {
55         Links = links;
56         Header = header;
57     }
58
59     /// <summary>
60     /// <para>
61     /// Gets the zero.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <returns>
66     /// <para>The uint</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override uint GetZero() => 0U;
71
72     /// <summary>
73     /// <para>
74     /// Determines whether this instance equal to zero.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="value">
79     /// <para>The value.</para>
80     /// <para></para>
81     /// </param>
82     /// <returns>
83     /// <para>The bool</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected override bool EqualToZero(uint value) => value == 0U;
88
89     /// <summary>
90     /// <para>
91     /// Determines whether this instance are equal.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="first">
96     /// <para>The first.</para>
97     /// <para></para>
98     /// </param>
99     /// <param name="second">
100    /// <para>The second.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The bool</para>

```

```

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

```

```

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

```

```

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

```



```

334     protected override uint Subtract(uint first, uint second) => first - second;
335
336     /// <summary>
337     /// <para>
338     /// Determines whether this instance first is to the left of second.
339     /// </para>
340     /// <para></para>
341     /// </summary>
342     /// <param name="first">
343     /// <para>The first.</para>
344     /// <para></para>
345     /// </param>
346     /// <param name="second">
347     /// <para>The second.</para>
348     /// <para></para>
349     /// </param>
350     /// <returns>
351     /// <para>The bool</para>
352     /// <para></para>
353     /// </returns>
354     [MethodImpl(MethodImplOptions.AggressiveInlining)]
355     protected override bool FirstIsToLeftOfSecond(uint first, uint second)
356     {
357         ref var firstLink = ref Links[first];
358         ref var secondLink = ref Links[second];
359         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
360             ↪ secondLink.Source, secondLink.Target);
361     }
362
363     /// <summary>
364     /// <para>
365     /// Determines whether this instance first is to the right of second.
366     /// </para>
367     /// <para></para>
368     /// </summary>
369     /// <param name="first">
370     /// <para>The first.</para>
371     /// <para></para>
372     /// </param>
373     /// <param name="second">
374     /// <para>The second.</para>
375     /// <para></para>
376     /// </param>
377     /// <returns>
378     /// <para>The bool</para>
379     /// <para></para>
380     /// </returns>
381     [MethodImpl(MethodImplOptions.AggressiveInlining)]
382     protected override bool FirstIsToTheRightOfSecond(uint first, uint second)
383     {
384         ref var firstLink = ref Links[first];
385         ref var secondLink = ref Links[second];
386         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
387             ↪ secondLink.Source, secondLink.Target);
388     }
389
390     /// <summary>
391     /// <para>
392     /// Gets the header reference.
393     /// </para>
394     /// <para></para>
395     /// </summary>
396     /// <returns>
397     /// <para>A ref links header of uint</para>
398     /// <para></para>
399     /// </returns>
400     [MethodImpl(MethodImplOptions.AggressiveInlining)]
401     protected override ref LinksHeader<uint> GetHeaderReference() => ref *Header;
402
403     /// <summary>
404     /// <para>
405     /// Gets the link reference using the specified link.
406     /// </para>
407     /// <para></para>
408     /// </summary>
409     /// <param name="link">
410     /// <para>The link.</para>
411     /// <para></para>

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 32 links size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksSizeBalancedTreeMethodsBase{uint}"/>
15     public unsafe abstract class UInt32LinksSizeBalancedTreeMethodsBase :
16         ↳ LinksSizeBalancedTreeMethodsBase<uint>
17     {
18         /// <summary>
19         /// <para>
20         /// The links.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         protected new readonly RawLink<uint>* Links;
25
26         /// <summary>
27         /// <para>
28         /// The header.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly LinksHeader<uint>* Header;
33
34         /// <summary>
35         /// <para>
36         /// Initializes a new <see cref="UInt32LinksSizeBalancedTreeMethodsBase"/> instance.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="constants">
41         /// <para>A constants.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="links">
45         /// <para>A links.</para>
46         /// <para></para>
47         /// </param>
48         /// <param name="header">
49         /// <para>A header.</para>
50         /// <para></para>
51         /// </param>
52         protected UInt32LinksSizeBalancedTreeMethodsBase(LinksConstants<uint> constants,
53             ↳ RawLink<uint>* links, LinksHeader<uint>* header)
54             : base(constants, (byte*)links, (byte*)header)
55         {
56             Links = links;
57             Header = header;
58         }
59
60         /// <summary>
61         /// <para>
62         /// Gets the zero.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <returns>
67         /// <para>The uint</para>
68         /// <para></para>
69         /// </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     /// <summary>
146     /// <para>
147     /// Determines whether this instance greater or equal than.
148     /// </para>
149     /// </summary>
150     /// <param name="first">
151     /// <para>The first.</para>
152     /// </param>
153     /// <param name="second">
154     /// <para>The second.</para>
155     /// </param>
156     /// <returns>
157     /// <para>The bool</para>
158     /// </returns>
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
161
162     /// <summary>
163     /// <para>
164     /// Determines whether this instance greater or equal than zero.
165     /// </para>
166     /// </summary>
167     /// <param name="value">
168     /// <para>The value.</para>
169     /// </param>
170     /// <returns>
171     /// <para>The bool</para>
172     /// </returns>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override bool GreaterOrEqualThanZero(uint value) => true; // value >= 0 is
175     ↪ always true for uint
176
177     /// <summary>
178     /// <para>
179     /// Determines whether this instance less or equal than zero.
180     /// </para>
181     /// </summary>
182     /// <param name="value">
183     /// <para>The value.</para>
184     /// </param>
185     /// <returns>
186     /// <para>The bool</para>
187     /// </returns>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override bool LessOrEqualThanZero(uint value) => value == 0U; // value is
190     ↪ always >= 0 for uint
191
192     /// <summary>
193     /// <para>
194     /// Determines whether this instance less or equal than.
195     /// </para>
196     /// </summary>
197     /// <param name="first">
198     /// <para>The first.</para>
199     /// </param>
200     /// <param name="second">
201     /// <para>The second.</para>
202     /// </param>
203     /// <returns>
204     /// <para>The bool</para>
205     /// </returns>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override bool LessOrEqualThan(uint first, uint second) => first <= second;

```

```

221
222     /// <summary>
223     /// <para>
224     /// Determines whether this instance less than zero.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <param name="value">
229     /// <para>The value.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool LessThanZero(uint value) => false; // value < 0 is always false
238     ↪ for uint
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance less than.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="first">
247     /// <para>The first.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="second">
251     /// <para>The second.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool LessThan(uint first, uint second) => first < second;
260
261     /// <summary>
262     /// <para>
263     /// Increments the value.
264     /// </para>
265     /// <para></para>
266     /// </summary>
267     /// <param name="value">
268     /// <para>The value.</para>
269     /// <para></para>
270     /// </param>
271     /// <returns>
272     /// <para>The uint</para>
273     /// <para></para>
274     /// </returns>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override uint Increment(uint value) => ++value;
277
278     /// <summary>
279     /// <para>
280     /// Decrements the value.
281     /// </para>
282     /// <para></para>
283     /// </summary>
284     /// <param name="value">
285     /// <para>The value.</para>
286     /// <para></para>
287     /// </param>
288     /// <returns>
289     /// <para>The uint</para>
290     /// <para></para>
291     /// </returns>
292     [MethodImpl(MethodImplOptions.AggressiveInlining)]
293     protected override uint Decrement(uint value) => --value;
294
295     /// <summary>
296     /// <para>
297     /// Adds the first.
298     /// </para>

```

```

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

```

```

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

```

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

```

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

```

```

30    /// <param name="header">
31    /// <para>A header.</para>
32    /// <para></para>
33    /// </param>
34    public UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
    ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
    ↪ header) { }

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

```



```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```

```

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

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

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

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

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

197     /// <summary>
198     /// <para>
199     /// Gets the base part value using the specified link.
200     /// </para>
201     /// <para></para>
202     /// </summary>

```

```

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

```

```

276         link.LeftAsSource = OU;
277         link.RightAsSource = OU;
278         link.SizeAsSource = OU;
279     }
280 }
281 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links targets recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods :
15         ↳ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↳ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// <para></para>
35         /// </param>
36         public UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
37             ↳ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
38             ↳ header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="node">
47         /// <para>The node.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The ref uint</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsTarget;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref uint</para>
69         /// <para></para>
70         /// </returns>

```

```

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

```



```

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

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links targets size balanced tree methods.
10    /// </para>
11    /// <para></para>

```

```

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

```

```

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

```

```

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

```

```

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

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

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Memory;
4  using Platform.Singletons;
5  using Platform.Data.Doublets.Memory.United.Generic;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     ↪ organizing the storage of links with addresses represented as <see cref="uint" />.</para>
14     /// <para>Представляет низкоуровневую 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>

```

```

31     /// </param>
32     [MethodImpl(MethodImplOptions.AggressiveInlining)]
33     public UInt32UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
34
35     /// <summary>
36     /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
37     /// ↪ минимальным шагом расширения базы данных.
38     /// </summary>
39     /// <param name="address">Полный путь к файлу базы данных.</param>
40     /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
41     /// ↪ байтах.</param>
42     [MethodImpl(MethodImplOptions.AggressiveInlining)]
43     public UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
44     ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
45     ↪ memoryReservationStep) { }
46
47     /// <summary>
48     /// <para>
49     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     /// <param name="memory">
54     /// <para>A memory.</para>
55     /// <para></para>
56     /// </param>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
59     ↪ DefaultLinksSizeStep) { }
60
61     /// <summary>
62     /// <para>
63     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
64     /// </para>
65     /// <para></para>
66     /// </summary>
67     /// <param name="memory">
68     /// <para>A memory.</para>
69     /// <para></para>
70     /// </param>
71     /// <param name="memoryReservationStep">
72     /// <para>A memory reservation step.</para>
73     /// <para></para>
74     /// </param>
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
77     ↪ memoryReservationStep) : this(memory, memoryReservationStep,
78     ↪ Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }
79
80     /// <summary>
81     /// <para>
82     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <param name="memory">
87     /// <para>A memory.</para>
88     /// <para></para>
89     /// </param>
90     /// <param name="memoryReservationStep">
91     /// <para>A memory reservation step.</para>
92     /// <para></para>
93     /// </param>
94     /// <param name="constants">
95     /// <para>A constants.</para>
96     /// <para></para>
97     /// </param>
98     /// <param name="indexTreeType">
99     /// <para>A index tree type.</para>
100    /// <para></para>
101    /// </param>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
104    ↪ memoryReservationStep, LinksConstants<uint> constants, IndexTreeType indexTreeType)
105    ↪ : base(memory, memoryReservationStep, constants)
106    {
107        if (indexTreeType == IndexTreeType.SizeBalancedTree)

```

```

99     {
100         _createSourceTreeMethods = () => new
101             ↳ UInt32LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
102         _createTargetTreeMethods = () => new
103             ↳ UInt32LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
104     }
105     else
106     {
107         _createSourceTreeMethods = () => new
108             ↳ UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
109             ↳ _header);
110         _createTargetTreeMethods = () => new
111             ↳ UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
112             ↳ _header);
113     }
114     Init(memory, memoryReservationStep);
115 }
116
117 /// <summary>
118 /// <para>
119 /// Sets the pointers using the specified memory.
120 /// </para>
121 /// <para></para>
122 /// </summary>
123 /// <param name="memory">
124 /// <para>The memory.</para>
125 /// <para></para>
126 /// </param>
127 [MethodImpl(MethodImplOptions.AggressiveInlining)]
128 protected override void SetPointers(IResizableDirectMemory memory)
129 {
130     _header = (LinksHeader<uint>*)memory.Pointer;
131     _links = (RawLink<uint>*)memory.Pointer;
132     SourcesTreeMethods = _createSourceTreeMethods();
133     TargetsTreeMethods = _createTargetTreeMethods();
134     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_links, _header);
135 }
136
137 /// <summary>
138 /// <para>
139 /// Resets the pointers.
140 /// </para>
141 /// <para></para>
142 /// </summary>
143 [MethodImpl(MethodImplOptions.AggressiveInlining)]
144 protected override void ResetPointers()
145 {
146     base.ResetPointers();
147     _links = null;
148     _header = null;
149 }
150
151 /// <summary>
152 /// <para>
153 /// Gets the header reference.
154 /// </para>
155 /// <para></para>
156 /// </summary>
157 /// <returns>
158 /// <para>A ref links header of uint</para>
159 /// <para></para>
160 /// </returns>
161 [MethodImpl(MethodImplOptions.AggressiveInlining)]
162 protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
163
164 /// <summary>
165 /// <para>
166 /// Gets the link reference using the specified link index.
167 /// </para>
168 /// <para></para>
169 /// </summary>
170 /// <param name="linkIndex">
171 /// <para>The link index.</para>
172 /// <para></para>
173 /// </param>
174 /// <returns>
175 /// <para>A ref raw link of uint</para>
176 /// <para></para>

```



```

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

```

```

248     /// <param name="second">
249     /// <para>The second.</para>
250     /// <para></para>
251     /// </param>
252     /// <returns>
253     /// <para>The bool</para>
254     /// <para></para>
255     /// </returns>
256     [MethodImpl(MethodImplOptions.AggressiveInlining)]
257     protected override bool GreaterThan(uint first, uint second) => first > second;
258
259     /// <summary>
260     /// <para>
261     /// Determines whether this instance greater or equal than.
262     /// </para>
263     /// <para></para>
264     /// </summary>
265     /// <param name="first">
266     /// <para>The first.</para>
267     /// <para></para>
268     /// </param>
269     /// <param name="second">
270     /// <para>The second.</para>
271     /// <para></para>
272     /// </param>
273     /// <returns>
274     /// <para>The bool</para>
275     /// <para></para>
276     /// </returns>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
279
280     /// <summary>
281     /// <para>
282     /// Gets the zero.
283     /// </para>
284     /// <para></para>
285     /// </summary>
286     /// <returns>
287     /// <para>The uint</para>
288     /// <para></para>
289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     protected override uint GetZero() => 0U;
292
293     /// <summary>
294     /// <para>
295     /// Gets the one.
296     /// </para>
297     /// <para></para>
298     /// </summary>
299     /// <returns>
300     /// <para>The uint</para>
301     /// <para></para>
302     /// </returns>
303     [MethodImpl(MethodImplOptions.AggressiveInlining)]
304     protected override uint GetOne() => 1U;
305
306     /// <summary>
307     /// <para>
308     /// Converts the to int 64 using the specified value.
309     /// </para>
310     /// <para></para>
311     /// </summary>
312     /// <param name="value">
313     /// <para>The value.</para>
314     /// <para></para>
315     /// </param>
316     /// <returns>
317     /// <para>The long</para>
318     /// <para></para>
319     /// </returns>
320     [MethodImpl(MethodImplOptions.AggressiveInlining)]
321     protected override long ConvertToInt64(uint value) => (long)value;
322
323     /// <summary>
324     /// <para>
325     /// Converts the to address using the specified value.

```

```

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

```

```

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

```

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

```

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

```

```

65     /// <returns>
66     /// <para>A ref links header of uint</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
71 }
72 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using static System.Runtime.CompilerServices.Unsafe;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.United.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 links avl balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksAvlBalancedTreeMethodsBase{ulong}" />
16     public unsafe abstract class UInt64LinksAvlBalancedTreeMethodsBase :
17     ↪ LinksAvlBalancedTreeMethodsBase<ulong>
18     {
19         /// <summary>
20         /// <para>
21         /// The links.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLink<ulong>* Links;
26         /// <summary>
27         /// <para>
28         /// The header.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly LinksHeader<ulong>* Header;
33
34         /// <summary>
35         /// <para>
36         /// Initializes a new <see cref="UInt64LinksAvlBalancedTreeMethodsBase" /> instance.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="constants">
41         /// <para>A constants.</para>
42         /// <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 UInt64LinksAvlBalancedTreeMethodsBase(LinksConstants<ulong> constants,
53     ↪ 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>

```

```

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

```

```

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

```

```

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

```

```

376    /// </param>
377    /// <returns>
378    /// <para>The bool</para>
379    /// <para></para>
380    /// </returns>
381    [MethodImpl(MethodImplOptions.AggressiveInlining)]
382    protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
383    {
384        ref var firstLink = ref Links[first];
385        ref var secondLink = ref Links[second];
386        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
387    }
388
389    /// <summary>
390    /// <para>
391    /// Gets the size value using the specified value.
392    /// </para>
393    /// <para></para>
394    /// </summary>
395    /// <param name="value">
396    /// <para>The value.</para>
397    /// <para></para>
398    /// </param>
399    /// <returns>
400    /// <para>The ulong</para>
401    /// <para></para>
402    /// </returns>
403    [MethodImpl(MethodImplOptions.AggressiveInlining)]
404    protected override ulong GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;
405
406    /// <summary>
407    /// <para>
408    /// Sets the size value using the specified stored value.
409    /// </para>
410    /// <para></para>
411    /// </summary>
412    /// <param name="storedValue">
413    /// <para>The stored value.</para>
414    /// <para></para>
415    /// </param>
416    /// <param name="size">
417    /// <para>The size.</para>
418    /// <para></para>
419    /// </param>
420    [MethodImpl(MethodImplOptions.AggressiveInlining)]
421    protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
        ↪ 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 /// </param>
467 /// <returns>
468 /// <para>The bool</para>
469 /// <para></para>
470 /// </returns>
471 [MethodImpl(MethodImplOptions.AggressiveInlining)]
472 protected override bool GetRightIsChildValue(ulong value) => (value & 8UL) >> 3 == 1UL;
473
474 /// <summary>
475 /// <para>
476 /// Sets the right is child value using the specified stored value.
477 /// </para>
478 /// <para></para>
479 /// </summary>
480 /// <param name="storedValue">
481 /// <para>The stored value.</para>
482 /// <para></para>
483 /// </param>
484 /// <param name="value">
485 /// <para>The value.</para>
486 /// <para></para>
487 /// </param>
488 [MethodImpl(MethodImplOptions.AggressiveInlining)]
489 protected override void SetRightIsChildValue(ref ulong storedValue, bool value) =>
490     ↳ storedValue = storedValue & 4294967287UL | (As<bool, byte>(ref value) & 1UL) << 3;
491
492 /// <summary>
493 /// <para>
494 /// Gets the balance value using the specified value.
495 /// </para>
496 /// <para></para>
497 /// </summary>
498 /// <param name="value">
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);

```

```

524
525     /// <summary>
526     /// <para>
527     /// Gets the header reference.
528     /// </para>
529     /// <para></para>
530     /// </summary>
531     /// <returns>
532     /// <para>A ref links header of ulong</para>
533     /// <para></para>
534     /// </returns>
535     [MethodImpl(MethodImplOptions.AggressiveInlining)]
536     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
537
538     /// <summary>
539     /// <para>
540     /// Gets the link reference using the specified link.
541     /// </para>
542     /// <para></para>
543     /// </summary>
544     /// <param name="link">
545     /// <para>The link.</para>
546     /// <para></para>
547     /// </param>
548     /// <returns>
549     /// <para>A ref raw link of ulong</para>
550     /// <para></para>
551     /// </returns>
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]
553     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
554 }
555 }

```

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

```

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

```

```

43     /// <para>A links.</para>
44     /// <para></para>
45     /// </param>
46     /// <param name="header">
47     /// <para>A header.</para>
48     /// <para></para>
49     /// </param>
50     protected UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
    ↳ constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
51         : base(constants, (byte*)links, (byte*)header)
52     {
53         Links = links;
54         Header = header;
55     }
56
57     /// <summary>
58     /// <para>
59     /// Gets the zero.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <returns>
64     /// <para>The ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ulong GetZero() => 0UL;
69
70     /// <summary>
71     /// <para>
72     /// Determines whether this instance equal to zero.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="value">
77     /// <para>The value.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The bool</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override bool EqualToZero(ulong value) => value == 0UL;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance are equal.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="first">
94     /// <para>The first.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="second">
98     /// <para>The second.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The bool</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override bool AreEqual(ulong first, ulong second) => first == second;
107
108    /// <summary>
109    /// <para>
110    /// Determines whether this instance greater than zero.
111    /// </para>
112    /// <para></para>
113    /// </summary>
114    /// <param name="value">
115    /// <para>The value.</para>
116    /// <para></para>
117    /// </param>
118    /// <returns>
119    /// <para>The bool</para>

```

```

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

```

```

197     /// </returns>
198     [MethodImpl(MethodImplOptions.AggressiveInlining)]
199     protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
    ↪ always >= 0 for ulong

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

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

```

```

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

```



```

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

```

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

```

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

```

```

7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 links size balanced tree methods base.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="LinksSizeBalancedTreeMethodsBase{ulong}" />
15    public unsafe abstract class UInt64LinksSizeBalancedTreeMethodsBase :
16    ↪ LinksSizeBalancedTreeMethodsBase<ulong>
17    {
18        /// <summary>
19        /// <para>
20        /// The links.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        protected new readonly RawLink<ulong>* Links;
25        /// <summary>
26        /// <para>
27        /// The header.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        protected new readonly LinksHeader<ulong>* Header;
32        /// <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() => 0UL;
69
70        /// <summary>
71        /// <para>
72        /// Determines whether this instance equal to zero.
73        /// </para>
74        /// <para></para>
75        /// </summary>
76        /// <param name="value">
77        /// <para>The value.</para>
78        /// <para></para>
79        /// </param>
80        /// <returns>
81        /// <para>The bool</para>
82        /// <para></para>

```

```

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

```

```

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

```

```

237     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
        ↳ 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>
381     [MethodImpl(MethodImplOptions.AggressiveInlining)]
382     protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
383     {
384         ref var firstLink = ref Links[first];
385         ref var secondLink = ref Links[second];
386         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
387             ↪ secondLink.Source, secondLink.Target);
388     }
389
390     /// <summary>
391     /// <para>

```

```

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         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
36             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37             ↳ { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="node">
46         /// <para>The node.</para>
47         /// <para></para>
48         /// </param>

```

```

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

```



```

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

```

```

269     /// <param name="node">
270     /// <para>The node.</para>
271     /// <para></para>
272     /// </param>
273     /// <param name="value">
274     /// <para>The value.</para>
275     /// <para></para>
276     /// </param>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
    ↪ Links[node].SizeAsSource, value);

279     /// <summary>
280     /// <para>
281     /// Gets the tree root.
282     /// </para>
283     /// <para></para>
284     /// </summary>
285     /// <returns>
286     /// <para>The ulong</para>
287     /// <para></para>
288     /// </returns>
289     [MethodImpl(MethodImplOptions.AggressiveInlining)]
290     protected override ulong GetTreeRoot() => Header->RootAsSource;

291     /// <summary>
292     /// <para>
293     /// Gets the base part value using the specified link.
294     /// </para>
295     /// <para></para>
296     /// </summary>
297     /// <param name="link">
298     /// <para>The link.</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 GetBasePartValue(ulong link) => Links[link].Source;

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>
329     /// <returns>
330     /// <para>The bool</para>
331     /// <para></para>
332     /// </returns>
333     [MethodImpl(MethodImplOptions.AggressiveInlining)]
334     protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
    ↪ ulong secondSource, ulong secondTarget)
335     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
    ↪ secondTarget);

336     /// <summary>
337     /// <para>
338     /// Determines whether this instance first is to the right of second.
339     /// </para>

```

```

344     /// <para></para>
345     /// </summary>
346     /// <param name="firstSource">
347     /// <para>The first source.</para>
348     /// <para></para>
349     /// </param>
350     /// <param name="firstTarget">
351     /// <para>The first target.</para>
352     /// <para></para>
353     /// </param>
354     /// <param name="secondSource">
355     /// <para>The second source.</para>
356     /// <para></para>
357     /// </param>
358     /// <param name="secondTarget">
359     /// <para>The second target.</para>
360     /// <para></para>
361     /// </param>
362     /// <returns>
363     /// <para>The bool</para>
364     /// <para></para>
365     /// </returns>
366     [MethodImpl(MethodImplOptions.AggressiveInlining)]
367     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
368         ↪ ulong secondSource, ulong secondTarget)
369         => firstSource > secondSource || (firstSource == secondSource && firstTarget >
370             ↪ secondTarget);
371
372     /// <summary>
373     /// <para>
374     /// <para>Clears the node using the specified node.
375     /// </para>
376     /// <para></para>
377     /// </summary>
378     /// <param name="node">
379     /// <para>The node.</para>
380     /// <para></para>
381     /// </param>
382     [MethodImpl(MethodImplOptions.AggressiveInlining)]
383     protected override void ClearNode(ulong node)
384     {
385         ref var link = ref Links[node];
386         link.LeftAsSource = OUL;
387         link.RightAsSource = OUL;
388         link.SizeAsSource = OUL;
389     }
390 }
391 }

```

## 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      /// <para>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         /// <para>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">

```

```

27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     public UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
        ↳ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
        ↳ links, header) { }

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

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

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

86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>

```

```

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

```

```

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

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

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links sources size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksSourcesSizeBalancedTreeMethods :
15         ↪ UInt64LinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64LinksSourcesSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
36             ↪ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37             ↪ { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>

```



```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links targets avl balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsAvlBalancedTreeMethods :
15         ↳ UInt64LinksAvlBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64LinksTargetsAvlBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt64LinksTargetsAvlBalancedTreeMethods(LinksConstants<ulong> constants,
36             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37         { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="node">
46         /// <para>The node.</para>
47         /// <para></para>
48         /// </param>
49         /// <returns>
50         /// <para>The ref ulong</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref ulong GetLeftReference(ulong node) => ref
55             ↳ Links[node].LeftAsTarget;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>

```

```

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) =>
222        ↪ GetRightIsChildValue(Links[node].SizeAsTarget);
223
224    /// <summary>
225    /// <para>
226    /// Sets the right is child using the specified node.
227    /// </para>
228    /// <para></para>
229    /// </summary>
230    /// <param name="node">
231    /// <para>The node.</para>
232    /// <para></para>
233    /// </param>
234    /// <param name="value">
235    /// <para>The value.</para>
236    /// <para></para>
237    /// </param>
238    [MethodImpl(MethodImplOptions.AggressiveInlining)]
239    protected override void SetRightIsChild(ulong node, bool value) =>
240        ↪ SetRightIsChildValue(ref Links[node].SizeAsTarget, value);
241
242    /// <summary>
243    /// <para>
244    /// Gets the balance using the specified node.
245    /// </para>
246    /// <para></para>
247    /// </summary>
248    /// <param name="node">
249    /// <para>The node.</para>
250    /// <para></para>
251    /// </param>
252    /// <returns>
253    /// <para>The sbyte</para>
254    /// <para></para>
255    /// </returns>
256    [MethodImpl(MethodImplOptions.AggressiveInlining)]
257    protected override sbyte GetBalance(ulong node) =>
258        ↪ GetBalanceValue(Links[node].SizeAsTarget);
259
260    /// <summary>
261    /// <para>
262    /// Sets the balance 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    /// <param name="value">
271    /// <para>The value.</para>
272    /// <para></para>
273    /// </param>
274    [MethodImpl(MethodImplOptions.AggressiveInlining)]
275    protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
276        ↪ Links[node].SizeAsTarget, value);
277
278    /// <summary>
279    /// <para>
280    /// Gets the tree root.
281    /// </para>
282    /// <para></para>
283    /// </summary>
284    /// <returns>

```

```

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

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

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

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

```



```

357     /// <para>The bool</para>
358     /// <para></para>
359     /// </returns>
360     [MethodImpl(MethodImplOptions.AggressiveInlining)]
361     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
362         ↪ ulong secondSource, ulong secondTarget)
363         => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
364             ↪ secondSource);
365
366     /// <summary>
367     /// <para>
368     /// Clears the node using the specified node.
369     /// </para>
370     /// <para></para>
371     /// </summary>
372     /// <param name="node">
373     /// <para>The node.</para>
374     /// <para></para>
375     /// </param>
376     [MethodImpl(MethodImplOptions.AggressiveInlining)]
377     protected override void ClearNode(ulong node)
378     {
379         ref var link = ref Links[node];
380         link.LeftAsTarget = OUL;
381         link.RightAsTarget = OUL;
382         link.SizeAsTarget = OUL;
383     }
384 }

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsSizeBalancedTreeMethods :
15         ↳ UInt64LinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64LinksTargetsSizeBalancedTreeMethods"/> instance.
20         /// <para></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 }

```

```

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

```

```

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

```

```

210     /// <para></para>
211     /// </param>
212     /// <param name="firstTarget">
213     /// <para>The first target.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="secondSource">
217     /// <para>The second source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
230     ↪     ulong secondSource, ulong secondTarget)
231     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
232     ↪     secondSource);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance first is to the right of second.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="firstSource">
241     /// <para>The first source.</para>
242     /// <para></para>
243     /// </param>
244     /// <param name="firstTarget">
245     /// <para>The first target.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="secondSource">
249     /// <para>The second source.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="secondTarget">
253     /// <para>The second target.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>
257     /// <para>The bool</para>
258     /// <para></para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
262     ↪     ulong secondSource, ulong secondTarget)
263     => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
264     ↪     secondSource);
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 }
285 }

```



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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Memory;
4  using Platform.Singletons;
5  using Platform.Data.Doublets.Memory.United.Generic;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     ///     ↪ organizing the storage of links with addresses represented as <see cref="ulong"
14     ///     ↪ />.</para>
15     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
16     ///     ↪ размером, для организации хранения связей с адресами представленными в виде <see
17     ///     ↪ cref="ulong"/>.</para>
18     /// </summary>
19     public unsafe class UInt64UnitedMemoryLinks : UnitedMemoryLinksBase<ulong>
20     {
21         private readonly Func<ILinksTreeMethods<ulong>> _createSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<ulong>> _createTargetTreeMethods;
23         private LinksHeader<ulong>* _header;
24         private RawLink<ulong>* _links;
25
26         /// <summary>
27         /// <para>
28         ///     Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="address">
33         ///     <para>A address.</para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt64UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
37
38         /// <summary>
39         ///     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 UInt64UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
47         ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
48         ↪ memoryReservationStep) { }
49
50         /// <summary>
51         /// <para>
52         ///     Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <param name="memory">
57         ///     <para>A memory.</para>
58         /// </param>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public UInt64UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
61         ↪ DefaultLinksSizeStep) { }
62
63         /// <summary>
64         /// <para>
65         ///     Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         /// <param name="memory">
70         ///     <para>A memory.</para>
71         /// </param>
72         /// <param name="memoryReservationStep">
73         ///     <para>A memory reservation step.</para>
74         /// </param>

```

```

69     /// </param>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep) : this(memory, memoryReservationStep,
        ↳ Default<LinksConstants<ulong>>.Instance, IndexTreeType.Default) { }

72
73     /// <summary>
74     /// <para>
75     /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <param name="memory">
80     /// <para>A memory.</para>
81     /// <para></para>
82     /// </param>
83     /// <param name="memoryReservationStep">
84     /// <para>A memory reservation step.</para>
85     /// <para></para>
86     /// </param>
87     /// <param name="constants">
88     /// <para>A constants.</para>
89     /// <para></para>
90     /// </param>
91     /// <param name="indexTreeType">
92     /// <para>A index tree type.</para>
93     /// <para></para>
94     /// </param>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep, LinksConstants<ulong> constants, IndexTreeType indexTreeType)
        ↳ : base(memory, memoryReservationStep, constants)
97     {
98         if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
99         {
100             _createSourceTreeMethods = () => new
                ↳ UInt64LinksSourcesAvlBalancedTreeMethods(Constants, _links, _header);
101             _createTargetTreeMethods = () => new
                ↳ UInt64LinksTargetsAvlBalancedTreeMethods(Constants, _links, _header);
102         }
103         else if (indexTreeType == IndexTreeType.SizeBalancedTree)
104         {
105             _createSourceTreeMethods = () => new
                ↳ UInt64LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
106             _createTargetTreeMethods = () => new
                ↳ UInt64LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
107         }
108         else
109         {
110             _createSourceTreeMethods = () => new
                ↳ UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
                ↳ _header);
111             _createTargetTreeMethods = () => new
                ↳ UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
                ↳ _header);
112         }
113         Init(memory, memoryReservationStep);
114     }
115
116     /// <summary>
117     /// <para>
118     /// Sets the pointers using the specified memory.
119     /// </para>
120     /// <para></para>
121     /// </summary>
122     /// <param name="memory">
123     /// <para>The memory.</para>
124     /// <para></para>
125     /// </param>
126     [MethodImpl(MethodImplOptions.AggressiveInlining)]
127     protected override void SetPointers(IResizableDirectMemory memory)
128     {
129         _header = (LinksHeader<ulong>*)memory.Pointer;
130         _links = (RawLink<ulong>*)memory.Pointer;
131         SourcesTreeMethods = _createSourceTreeMethods();
132         TargetsTreeMethods = _createTargetTreeMethods();
133         UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);

```

```

134     }
135
136     /// <summary>
137     /// <para>
138     /// Resets the pointers.
139     /// </para>
140     /// <para></para>
141     /// </summary>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     protected override void ResetPointers()
144     {
145         base.ResetPointers();
146         _links = null;
147         _header = null;
148     }
149
150     /// <summary>
151     /// <para>
152     /// Gets the header reference.
153     /// </para>
154     /// <para></para>
155     /// </summary>
156     /// <returns>
157     /// <para>A ref links header of ulong</para>
158     /// <para></para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
162
163     /// <summary>
164     /// <para>
165     /// Gets the link reference using the specified link index.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="linkIndex">
170     /// <para>The link index.</para>
171     /// <para></para>
172     /// </param>
173     /// <returns>
174     /// <para>A ref raw link of ulong</para>
175     /// <para></para>
176     /// </returns>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
179         ↪ _links[linkIndex];
180
181     /// <summary>
182     /// <para>
183     /// Determines whether this instance are equal.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="first">
188     /// <para>The first.</para>
189     /// <para></para>
190     /// </param>
191     /// <param name="second">
192     /// <para>The second.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The bool</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override bool AreEqual(ulong first, ulong second) => first == second;
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance less than.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="first">
209     /// <para>The first.</para>
210     /// <para></para>
211     /// </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     }

```

```

26     /// </summary>
27     /// <param name="links">
28     /// <para>A links.</para>
29     /// <para></para>
30     /// </param>
31     [MethodImpl(MethodImplOptions.AggressiveInlining)]
32     public PropertiesOperator(ILinks<TLinkAddress> links) : base(links) { }
33
34     /// <summary>
35     /// <para>
36     /// Gets the value using the specified object.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     /// <param name="@object">
41     /// <para>The object.</para>
42     /// <para></para>
43     /// </param>
44     /// <param name="property">
45     /// <para>The property.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public TLinkAddress GetValue(TLinkAddress @object, TLinkAddress property)
54     {
55         var links = _links;
56         var objectProperty = links.SearchOrDefault(@object, property);
57         if (_equalityComparer.Equals(objectProperty, default))
58         {
59             return default;
60         }
61         var constants = links.Constants;
62         var any = constants.Any;
63         var query = new Link<TLinkAddress>(any, objectProperty, any);
64         var valueLink = links.SingleOrDefault(query);
65         if (valueLink == null)
66         {
67             return default;
68         }
69         return links.GetTarget(valueLink[constants.IndexPart]);
70     }
71
72     /// <summary>
73     /// <para>
74     /// Sets the value using the specified object.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="@object">
79     /// <para>The object.</para>
80     /// <para></para>
81     /// </param>
82     /// <param name="property">
83     /// <para>The property.</para>
84     /// <para></para>
85     /// </param>
86     /// <param name="value">
87     /// <para>The value.</para>
88     /// <para></para>
89     /// </param>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public void SetValue(TLinkAddress @object, TLinkAddress property, TLinkAddress value)
92     {
93         var links = _links;
94         var objectProperty = links.GetOrCreate(@object, property);
95         links.DeleteMany(links.AllIndices(links.Constants.Any, objectProperty));
96         links.GetOrCreate(objectProperty, value);
97     }
98 }
99 }

```

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

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;

```



```

4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.PropertyOperators
8 {
9     /// <summary>
10     /// <para>
11     /// Represents the property operator.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLinkAddress}"/>
16     /// <seealso cref="IProperty{TLinkAddress, TLinkAddress}"/>
17     public class PropertyOperator<TLinkAddress> : LinksOperatorBase<TLinkAddress>,
18         ↳ IProperty<TLinkAddress, TLinkAddress>
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         /// </summary>
30         /// <param name="links">
31         /// <para>A links.</para>
32         /// </param>
33         /// <param name="propertyMarker">
34         /// <para>A property marker.</para>
35         /// </param>
36         /// <param name="propertyValueMarker">
37         /// <para>A property value marker.</para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public PropertyOperator(ILinks<TLinkAddress> links, TLinkAddress propertyMarker,
41             ↳ TLinkAddress propertyValueMarker) : base(links)
42         {
43             _propertyMarker = propertyMarker;
44             _propertyValueMarker = propertyValueMarker;
45         }
46
47         /// <summary>
48         /// <para>
49         /// Gets the link.
50         /// </para>
51         /// </summary>
52         /// <param name="link">
53         /// <para>The link.</para>
54         /// </param>
55         /// <returns>
56         /// <para>The link</para>
57         /// </returns>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public TLinkAddress Get(TLinkAddress link)
60         {
61             var property = _links.SearchOrDefault(link, _propertyMarker);
62             return GetValue(GetContainer(property));
63         }
64
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         private TLinkAddress GetContainer(TLinkAddress property)
67         {
68             var valueContainer = default(TLinkAddress);
69             if (_equalityComparer.Equals(property, default))
70             {
71                 return valueContainer;
72             }
73             var links = _links;
74             var constants = links.Constants;
75             var continueConstant = constants.Continue;
76             var breakConstant = constants.Break;
77
78
79

```

```

80     var anyConstant = constants.Any;
81     var query = new Link<TLinkAddress>(anyConstant, property, anyConstant);
82     links.Each(candidate =>
83     {
84         var candidateTarget = links.GetTarget(candidate);
85         var valueTarget = links.GetTarget(candidateTarget);
86         if (_equalityComparer.Equals(valueTarget, _propertyValueMarker))
87         {
88             valueContainer = links.GetIndex(candidate);
89             return breakConstant;
90         }
91         return countinueConstant;
92     }, query);
93     return valueContainer;
94 }
95 [MethodImpl(MethodImplOptions.AggressiveInlining)]
96 private TLinkAddress GetValue(TLinkAddress container) =>
97     ↪ _equalityComparer.Equals(container, default) ? default : _links.GetTarget(container);
98
99 /// <summary>
100 /// <para>
101 /// Sets the link.
102 /// </para>
103 /// </summary>
104 /// <param name="link">
105 /// <para>The link.</para>
106 /// </param>
107 /// <param name="value">
108 /// <para>The value.</para>
109 /// </param>
110 [MethodImpl(MethodImplOptions.AggressiveInlining)]
111 public void Set(TLinkAddress link, TLinkAddress value)
112 {
113     var links = _links;
114     var property = links.GetOrCreate(link, _propertyMarker);
115     var container = GetContainer(property);
116     if (_equalityComparer.Equals(container, default))
117     {
118         links.GetOrCreate(property, value);
119     }
120     else
121     {
122         links.Update(container, property, value);
123     }
124 }
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     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLinkAddress}"/>
16     /// <seealso cref="IStack{TLinkAddress}"/>
17     public class Stack<TLinkAddress> : LinksOperatorBase<TLinkAddress>, IStack<TLinkAddress>
18     {
19         private static readonly EqualityComparer<TLinkAddress> _equalityComparer =
20             ↪ EqualityComparer<TLinkAddress>.Default;
21         private readonly TLinkAddress _stack;
22
23         /// <summary>
24         /// <para>
25         /// Gets the is empty value.
26         /// </para>

```

```

27     /// <para></para>
28     /// </summary>
29     public bool IsEmpty
30     {
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         get => _equalityComparer.Equals(Peek(), _stack);
33     }
34
35     /// <summary>
36     /// <para>
37     /// Initializes a new <see cref="Stack"/> instance.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     /// <param name="links">
42     /// <para>A links.</para>
43     /// <para></para>
44     /// </param>
45     /// <param name="stack">
46     /// <para>A stack.</para>
47     /// <para></para>
48     /// </param>
49     [MethodImpl(MethodImplOptions.AggressiveInlining)]
50     public Stack(ILinks<TLinkAddress> links, TLinkAddress stack) : base(links) => _stack =
51         ↪ stack;
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     private TLinkAddress GetStackMarker() => _links.GetSource(_stack);
54     [MethodImpl(MethodImplOptions.AggressiveInlining)]
55     private TLinkAddress GetTop() => _links.GetTarget(_stack);
56
57     /// <summary>
58     /// <para>
59     /// Peeks this instance.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <returns>
64     /// <para>The link</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public TLinkAddress Peek() => _links.GetTarget(GetTop());
69
70     /// <summary>
71     /// <para>
72     /// Pops this instance.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <returns>
77     /// <para>The element.</para>
78     /// <para></para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public TLinkAddress Pop()
82     {
83         var element = Peek();
84         if (!_equalityComparer.Equals(element, _stack))
85         {
86             var top = GetTop();
87             var previousTop = _links.GetSource(top);
88             _links.Update(_stack, GetStackMarker(), previousTop);
89             _links.Delete(top);
90         }
91         return element;
92     }
93
94     /// <summary>
95     /// <para>
96     /// Pushes the element.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="element">
101    /// <para>The element.</para>
102    /// <para></para>
103    /// </param>
104    [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>

```

```

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

```

```

177     public TLinkAddress Update(IList<TLinkAddress>? restriction, IList<TLinkAddress>?
        ↳ substitution, WriteHandler<TLinkAddress>? handler) =>
        ↳ SyncRoot.ExecuteWriteOperation(restriction, substitution, handler, Unsync.Update);
178
179     /// <summary>
180     /// <para>
181     /// Deletes the substitution.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="restriction">
186     /// <para>The substitution.</para>
187     /// <para></para>
188     /// </param>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     public TLinkAddress Delete(IList<TLinkAddress>? restriction, WriteHandler<TLinkAddress>?
        ↳ handler) => SyncRoot.ExecuteWriteOperation(restriction, handler, Unsync.Delete);
191
192     //public T Trigger(IList<T> restriction, Func<IList<T>, IList<T>, T> matchedHandler,
        ↳ IList<T> substitution, Func<IList<T>, IList<T>, T> substitutedHandler)
193     //{
194     //    if (restriction != null && substitution != null &&
        ↳ !substitution.EqualTo(restriction))
195     //        return SyncRoot.ExecuteWriteOperation(restriction, matchedHandler,
        ↳ substitution, substitutedHandler, Unsync.Trigger);
196
197     //    return SyncRoot.ExecuteReadOperation(restriction, matchedHandler, substitution,
        ↳ substitutedHandler, Unsync.Trigger);
198     //}
199 }
200 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Singletons;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the int 64 links extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class UInt64LinksExtensions
18     {
19         /// <summary>
20         /// <para>
21         /// The instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public static readonly LinksConstants<ulong> Constants =
            ↳ Default<LinksConstants<ulong>>.Instance;
26
27         /// <summary>
28         /// <para>
29         /// Determines whether any link is any.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="links">
34         /// <para>The links.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="sequence">
38         /// <para>The sequence.</para>
39         /// <para></para>
40         /// </param>
41         /// <returns>
42         /// <para>The bool</para>
43         /// <para></para>
44         /// </returns>

```

```

45 [MethodImpl(MethodImplOptions.AggressiveInlining)]
46 public static bool AnyLinkIsAny(this ILinks<ulong> links, params ulong[] sequence)
47 {
48     if (sequence == null)
49     {
50         return false;
51     }
52     var constants = links.Constants;
53     for (var i = 0; i < sequence.Length; i++)
54     {
55         if (sequence[i] == constants.Any)
56         {
57             return true;
58         }
59     }
60     return false;
61 }
62
63 /// <summary>
64 /// <para>
65 /// Formats the structure using the specified links.
66 /// </para>
67 /// <para></para>
68 /// </summary>
69 /// <param name="links">
70 /// <para>The links.</para>
71 /// <para></para>
72 /// </param>
73 /// <param name="linkIndex">
74 /// <para>The link index.</para>
75 /// <para></para>
76 /// </param>
77 /// <param name="isElement">
78 /// <para>The is element.</para>
79 /// <para></para>
80 /// </param>
81 /// <param name="renderIndex">
82 /// <para>The render index.</para>
83 /// <para></para>
84 /// </param>
85 /// <param name="renderDebug">
86 /// <para>The render debug.</para>
87 /// <para></para>
88 /// </param>
89 /// <returns>
90 /// <para>The string</para>
91 /// <para></para>
92 /// </returns>
93 [MethodImpl(MethodImplOptions.AggressiveInlining)]
94 public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
95     ↪ Func<Link<ulong>, bool> isElement, bool renderIndex = false, bool renderDebug =
96     ↪ false)
97 {
98     var sb = new StringBuilder();
99     var visited = new HashSet<ulong>();
100     links.AppendStructure(sb, visited, linkIndex, isElement, (innerSb, link) =>
101         ↪ innerSb.Append(link.Index), renderIndex, renderDebug);
102     return sb.ToString();
103 }
104
105 /// <summary>
106 /// <para>
107 /// Formats the structure using the specified links.
108 /// </para>
109 /// <para></para>
110 /// </summary>
111 /// <param name="links">
112 /// <para>The links.</para>
113 /// <para></para>
114 /// </param>
115 /// <param name="linkIndex">
116 /// <para>The link index.</para>
117 /// <para></para>
118 /// </param>
119 /// <param name="isElement">
120 /// <para>The is element.</para>
121 /// <para></para>
122 /// </param>

```



```

120     /// <param name="appendElement">
121     /// <para>The append element.</para>
122     /// <para></para>
123     /// </param>
124     /// <param name="renderIndex">
125     /// <para>The render index.</para>
126     /// <para></para>
127     /// </param>
128     /// <param name="renderDebug">
129     /// <para>The render debug.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The string</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
    ↪ Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
    ↪ bool renderIndex = false, bool renderDebug = false)
138     {
139         var sb = new StringBuilder();
140         var visited = new HashSet<ulong>();
141         links.AppendStructure(sb, visited, linkIndex, isElement, appendElement, renderIndex,
    ↪ renderDebug);
142         return sb.ToString();
143     }
144
145     /// <summary>
146     /// <para>
147     /// Appends the structure using the specified links.
148     /// </para>
149     /// <para></para>
150     /// </summary>
151     /// <param name="links">
152     /// <para>The links.</para>
153     /// <para></para>
154     /// </param>
155     /// <param name="sb">
156     /// <para>The sb.</para>
157     /// <para></para>
158     /// </param>
159     /// <param name="visited">
160     /// <para>The visited.</para>
161     /// <para></para>
162     /// </param>
163     /// <param name="linkIndex">
164     /// <para>The link index.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="isElement">
168     /// <para>The is element.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="appendElement">
172     /// <para>The append element.</para>
173     /// <para></para>
174     /// </param>
175     /// <param name="renderIndex">
176     /// <para>The render index.</para>
177     /// <para></para>
178     /// </param>
179     /// <param name="renderDebug">
180     /// <para>The render debug.</para>
181     /// <para></para>
182     /// </param>
183     /// <exception cref="ArgumentNullException">
184     /// <para></para>
185     /// <para></para>
186     /// </exception>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     public static void AppendStructure(this ILinks<ulong> links, StringBuilder sb,
    ↪ HashSet<ulong> visited, ulong linkIndex, Func<Link<ulong>, bool> isElement,
    ↪ Action<StringBuilder, Link<ulong>> appendElement, bool renderIndex = false, bool
    ↪ renderDebug = false)
189     {
190         if (sb == null)

```

```

191 {
192     throw new ArgumentNullException(nameof(sb));
193 }
194 if (linkIndex == Constants.Null || linkIndex == Constants.Any || linkIndex ==
↳ Constants.Itself)
195 {
196     return;
197 }
198 if (links.Exists(linkIndex))
199 {
200     if (visited.Add(linkIndex))
201     {
202         sb.Append('(');
203         var link = new Link<ulong>(links.GetLink(linkIndex));
204         if (renderIndex)
205         {
206             sb.Append(link.Index);
207             sb.Append(':');
208         }
209         if (link.Source == link.Index)
210         {
211             sb.Append(link.Index);
212         }
213         else
214         {
215             var source = new Link<ulong>(links.GetLink(link.Source));
216             if (isElement(source))
217             {
218                 appendElement(sb, source);
219             }
220             else
221             {
222                 links.AppendStructure(sb, visited, source.Index, isElement,
↳ appendElement, renderIndex);
223             }
224         }
225         sb.Append(' ');
226         if (link.Target == link.Index)
227         {
228             sb.Append(link.Index);
229         }
230         else
231         {
232             var target = new Link<ulong>(links.GetLink(link.Target));
233             if (isElement(target))
234             {
235                 appendElement(sb, target);
236             }
237             else
238             {
239                 links.AppendStructure(sb, visited, target.Index, isElement,
↳ appendElement, renderIndex);
240             }
241         }
242         sb.Append(')');
243     }
244     else
245     {
246         if (renderDebug)
247         {
248             sb.Append('*');
249         }
250         sb.Append(linkIndex);
251     }
252 }
253 else
254 {
255     if (renderDebug)
256     {
257         sb.Append('~');
258     }
259     sb.Append(linkIndex);
260 }
261 }
262 }
263 }

```

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

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.IO;
5  using System.Runtime.CompilerServices;
6  using System.Threading;
7  using System.Threading.Tasks;
8  using Platform.Disposables;
9  using Platform.Timestamps;
10 using Platform.Unsafe;
11 using Platform.IO;
12 using Platform.Data.Doublets.Decorators;
13 using Platform.Delegates;
14 using Platform.Exceptions;
15
16 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18 namespace Platform.Data.Doublets
19 {
20     /// <summary>
21     /// <para>
22     /// Represents the int 64 links transactions layer.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <seealso cref="LinksDisposableDecoratorBase{ulong}" />
27     public class UInt64LinksTransactionsLayer : LinksDisposableDecoratorBase<ulong> //-V3073
28     {
29         /// <remarks>
30         /// Альтернативные варианты хранения трансформации (элемента транзакции):
31         ///
32         /// private enum TransitionType
33         /// {
34         ///     Creation,
35         ///     UpdateOf,
36         ///     UpdateTo,
37         ///     Deletion
38         /// }
39         ///
40         /// private struct Transition
41         /// {
42         ///     public ulong TransactionId;
43         ///     public UniqueTimestamp Timestamp;
44         ///     public TransactionItemType Type;
45         ///     public Link Source;
46         ///     public Link Linker;
47         ///     public Link Target;
48         /// }
49         ///
50         /// Или
51         ///
52         /// public struct TransitionHeader
53         /// {
54         ///     public ulong TransactionIdCombined;
55         ///     public ulong TimestampCombined;
56         ///
57         ///     public ulong TransactionId
58         ///     {
59         ///         get
60         ///         {
61         ///             return (ulong) mask & TransactionIdCombined;
62         ///         }
63         ///     }
64         ///
65         ///     public UniqueTimestamp Timestamp
66         ///     {
67         ///         get
68         ///         {
69         ///             return (UniqueTimestamp)mask & TransactionIdCombined;
70         ///         }
71         ///     }
72         ///
73         ///     public TransactionItemType Type
74         ///     {
75         ///         get
76         ///         {
77         ///             // Использовать по одному биту из TransactionId и Timestamp,
78         ///             // для значения в 2 бита, которое представляет тип операции

```

```

79         throw new NotImplementedException();
80     }
81 }
82 }
83
84 /// private struct Transition
85 {
86     public TransitionHeader Header;
87     public Link Source;
88     public Link Linker;
89     public Link Target;
90 }
91
92 /// </remarks>
93 public struct Transition : IEquatable<Transition>
94 {
95     /// <summary>
96     /// <para>
97     /// The size.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    public static readonly long Size = Structure<Transition>.Size;
102
103    /// <summary>
104    /// <para>
105    /// The transaction id.
106    /// </para>
107    /// <para></para>
108    /// </summary>
109    public readonly ulong TransactionId;
110
111    /// <summary>
112    /// <para>
113    /// The before.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    public readonly Link<ulong> Before;
118
119    /// <summary>
120    /// <para>
121    /// The after.
122    /// </para>
123    /// <para></para>
124    /// </summary>
125    public readonly Link<ulong> After;
126
127    /// <summary>
128    /// <para>
129    /// The timestamp.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    public readonly Timestamp Timestamp;
134
135    /// <summary>
136    /// <para>
137    /// Initializes a new <see cref="Transition"/> instance.
138    /// </para>
139    /// <para></para>
140    /// </summary>
141    /// <param name="uniqueTimestampFactory">
142    /// <para>A unique timestamp factory.</para>
143    /// <para></para>
144    /// </param>
145    /// <param name="transactionId">
146    /// <para>A transaction id.</para>
147    /// <para></para>
148    /// </param>
149    /// <param name="before">
150    /// <para>A before.</para>
151    /// <para></para>
152    /// </param>
153    /// <param name="after">
154    /// <para>A after.</para>
155    /// <para></para>
156    /// </param>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
159        ↪ transactionId, Link<ulong> before, Link<ulong> after)

```

```

156     {
157         TransactionId = transactionId;
158         Before = before;
159         After = after;
160         Timestamp = uniqueTimestampFactory.Create();
161     }
162
163     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
    ↪ transactionId, IList<ulong> before, IList<ulong> after) :
    ↪ this(uniqueTimestampFactory, transactionId, new Link<ulong>(before), new
    ↪ Link<ulong>(after)) { }
164
165     /// <summary>
166     /// <para>
167     /// Initializes a new <see cref="Transition"/> instance.
168     /// </para>
169     /// <para></para>
170     /// </summary>
171     /// <param name="uniqueTimestampFactory">
172     /// <para>A unique timestamp factory.</para>
173     /// <para></para>
174     /// </param>
175     /// <param name="transactionId">
176     /// <para>A transaction id.</para>
177     /// <para></para>
178     /// </param>
179     /// <param name="before">
180     /// <para>A before.</para>
181     /// <para></para>
182     /// </param>
183     [MethodImpl(MethodImplOptions.AggressiveInlining)]
184     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
    ↪ transactionId, Link<ulong> before) : this(uniqueTimestampFactory, transactionId,
    ↪ before, default) { }
185
186     /// <summary>
187     /// <para>
188     /// Initializes a new <see cref="Transition"/> instance.
189     /// </para>
190     /// <para></para>
191     /// </summary>
192     /// <param name="uniqueTimestampFactory">
193     /// <para>A unique timestamp factory.</para>
194     /// <para></para>
195     /// </param>
196     /// <param name="transactionId">
197     /// <para>A transaction id.</para>
198     /// <para></para>
199     /// </param>
200     [MethodImpl(MethodImplOptions.AggressiveInlining)]
201     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
    ↪ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
    ↪ }
202
203     /// <summary>
204     /// <para>
205     /// Returns the string.
206     /// </para>
207     /// <para></para>
208     /// </summary>
209     /// <returns>
210     /// <para>The string</para>
211     /// <para></para>
212     /// </returns>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     public override string ToString() => $"{Timestamp} {TransactionId}: {Before} =>
    ↪ {After}";
215
216     /// <summary>
217     /// <para>
218     /// Determines whether this instance equals.
219     /// </para>
220     /// <para></para>
221     /// </summary>
222     /// <param name="obj">
223     /// <para>The obj.</para>
224     /// <para></para>
225     /// </param>

```

```

226     /// <returns>
227     /// <para>The bool</para>
228     /// <para></para>
229     /// </returns>
230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
231     public override bool Equals(object obj) => obj is Transition transition ?
        ↳ Equals(transition) : false;

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

245
246     /// <summary>
247     /// <para>
248     /// Determines whether this instance equals.
249     /// </para>
250     /// <para></para>
251     /// </summary>
252     /// <param name="other">
253     /// <para>The other.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>
257     /// <para>The bool</para>
258     /// <para></para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     public bool Equals(Transition other) => TransactionId == other.TransactionId &&
        ↳ Before == other.Before && After == other.After && Timestamp == other.Timestamp;

262
263     [MethodImpl(MethodImplOptions.AggressiveInlining)]
264     public static bool operator ==(Transition left, Transition right) =>
        ↳ left.Equals(right);

265
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     public static bool operator !=(Transition left, Transition right) => !(left ==
        ↳ right);

268 }

269
270 /// <remarks>
271 /// Другие варианты реализации транзакций (атомарности):
272 /// 1. Разделение хранения значения связи ((Source Target) или (Source Linker
    ↳ Target)) и индексов.
273 /// 2. Хранение трансформаций/операций в отдельном хранилище Links, но дополнительно
    ↳ потребуется решить вопрос
274 /// со ссылками на внешние идентификаторы, или как-то иначе решить вопрос с
    ↳ пересечениями идентификаторов.
275 ///
276 /// Где хранить промежуточный список транзакций?
277 ///
278 /// В оперативной памяти:
279 /// Минусы:
280 /// 1. Может усложнить систему, если она будет функционировать самостоятельно,
281 /// так как нужно отдельно выделять память под список трансформаций.
282 /// 2. Выделенной оперативной памяти может не хватить, в том случае,
283 /// если транзакция использует слишком много трансформаций.
284 /// -> Можно использовать жёсткий диск для слишком длинных транзакций.
285 /// -> Максимальный размер списка трансформаций можно ограничить / задать
    ↳ константой.
286 /// 3. При подтверждении транзакции (Commit) все трансформации записываются разом
    ↳ создавая задержку.
287 ///
288 /// На жёстком диске:
289 /// Минусы:
290 /// 1. Длительный отклик, на запись каждой трансформации.
291 /// 2. Лог транзакций дополнительно наполняется отменёнными транзакциями.
292 /// -> Это может решаться упаковкой/исключением дублирующих операций.
293 /// -> Также это может решаться тем, что короткие транзакции вообще

```

```

294     ///         не будут записываться в случае отката.
295     ///         3. Перед тем как выполнять отмену операций транзакции нужно дождаться пока все
    → операции (трансформации)
296     ///         будут записаны в лог.
297     ///
298     /// </remarks>
299     public class Transaction : DisposableBase
300     {
301         private readonly Queue<Transition> _transitions;
302         private readonly UInt64LinksTransactionsLayer _layer;
303         /// <summary>
304         /// <para>
305         /// Gets or sets the is committed value.
306         /// </para>
307         /// <para></para>
308         /// </summary>
309         public bool IsCommitted { get; private set; }
310         /// <summary>
311         /// <para>
312         /// Gets or sets the is reverted value.
313         /// </para>
314         /// <para></para>
315         /// </summary>
316         public bool IsReverted { get; private set; }
317
318         /// <summary>
319         /// <para>
320         /// Initializes a new <see cref="Transaction"/> instance.
321         /// </para>
322         /// <para></para>
323         /// </summary>
324         /// <param name="layer">
325         /// <para>A layer.</para>
326         /// <para></para>
327         /// </param>
328         /// <exception cref="NotSupportedException">
329         /// <para>Nested transactions not supported.</para>
330         /// <para></para>
331         /// </exception>
332         [MethodImpl(MethodImplOptions.AggressiveInlining)]
333         public Transaction(UInt64LinksTransactionsLayer layer)
334         {
335             _layer = layer;
336             if (_layer._currentTransactionId != 0)
337             {
338                 throw new NotSupportedException("Nested transactions not supported.");
339             }
340             IsCommitted = false;
341             IsReverted = false;
342             _transitions = new Queue<Transition>();
343             SetCurrentTransaction(layer, this);
344         }
345
346         /// <summary>
347         /// <para>
348         /// Commits this instance.
349         /// </para>
350         /// <para></para>
351         /// </summary>
352         [MethodImpl(MethodImplOptions.AggressiveInlining)]
353         public void Commit()
354         {
355             EnsureTransactionAllowsWriteOperations(this);
356             while (_transitions.Count > 0)
357             {
358                 var transition = _transitions.Dequeue();
359                 _layer._transitions.Enqueue(transition);
360             }
361             _layer._lastCommittedTransactionId = _layer._currentTransactionId;
362             IsCommitted = true;
363         }
364         [MethodImpl(MethodImplOptions.AggressiveInlining)]
365         private void Revert()
366         {
367             EnsureTransactionAllowsWriteOperations(this);
368             var transitionsToRevert = new Transition[_transitions.Count];
369             _transitions.CopyTo(transitionsToRevert, 0);
370             for (var i = transitionsToRevert.Length - 1; i >= 0; i--)

```

```

371     {
372         _layer.RevertTransition(transitionsToRevert[i]);
373     }
374     IsReverted = true;
375 }
376
377 /// <summary>
378 /// <para>
379 /// Sets the current transaction using the specified layer.
380 /// </para>
381 /// <para></para>
382 /// </summary>
383 /// <param name="layer">
384 /// <para>The layer.</para>
385 /// <para></para>
386 /// </param>
387 /// <param name="transaction">
388 /// <para>The transaction.</para>
389 /// <para></para>
390 /// </param>
391 [MethodImpl(MethodImplOptions.AggressiveInlining)]
392 public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
    ↪ Transaction transaction)
393 {
394     layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
395     layer._currentTransactionTransitions = transaction._transitions;
396     layer._currentTransaction = transaction;
397 }
398
399 /// <summary>
400 /// <para>
401 /// Ensures the transaction allows write operations using the specified transaction.
402 /// </para>
403 /// <para></para>
404 /// </summary>
405 /// <param name="transaction">
406 /// <para>The transaction.</para>
407 /// <para></para>
408 /// </param>
409 /// <exception cref="InvalidOperationException">
410 /// <para>Transation is committed.</para>
411 /// <para></para>
412 /// </exception>
413 /// <exception cref="InvalidOperationException">
414 /// <para>Transation is reverted.</para>
415 /// <para></para>
416 /// </exception>
417 [MethodImpl(MethodImplOptions.AggressiveInlining)]
418 public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
419 {
420     if (transaction.IsReverted)
421     {
422         throw new InvalidOperationException("Transation is reverted.");
423     }
424     if (transaction.IsCommitted)
425     {
426         throw new InvalidOperationException("Transation is committed.");
427     }
428 }
429
430 /// <summary>
431 /// <para>
432 /// Disposes the manual.
433 /// </para>
434 /// <para></para>
435 /// </summary>
436 /// <param name="manual">
437 /// <para>The manual.</para>
438 /// <para></para>
439 /// </param>
440 /// <param name="wasDisposed">
441 /// <para>The was disposed.</para>
442 /// <para></para>
443 /// </param>
444 [MethodImpl(MethodImplOptions.AggressiveInlining)]
445 protected override void Dispose(bool manual, bool wasDisposed)
446 {
447     if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)

```



```

448         {
449             if (!IsCommitted && !IsReverted)
450             {
451                 Revert();
452             }
453             _layer.ResetCurrentTransation();
454         }
455     }
456 }
457
458 /// <summary>
459 /// <para>
460 /// The from seconds.
461 /// </para>
462 /// <para></para>
463 /// </summary>
464 public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
465 private readonly string _logAddress;
466 private readonly FileStream _log;
467 private readonly Queue<Transition> _transitions;
468 private readonly UniqueTimestampFactory _uniqueTimestampFactory;
469 private Task _transitionsPusher;
470 private Transition _lastCommittedTransition;
471 private ulong _currentTransactionId;
472 private Queue<Transition> _currentTransactionTransitions;
473 private Transaction _currentTransaction;
474 private ulong _lastCommittedTransactionId;
475
476 /// <summary>
477 /// <para>
478 /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.
479 /// </para>
480 /// <para></para>
481 /// </summary>
482 /// <param name="links">
483 /// <para>A links.</para>
484 /// <para></para>
485 /// </param>
486 /// <param name="logAddress">
487 /// <para>A log address.</para>
488 /// <para></para>
489 /// </param>
490 /// <exception cref="ArgumentNullException">
491 /// <para></para>
492 /// <para></para>
493 /// </exception>
494 /// <exception cref="NotSupportedException">
495 /// <para>Database is damaged, autorecovery is not supported yet.</para>
496 /// <para></para>
497 /// </exception>
498 [MethodImpl(MethodImplOptions.AggressiveInlining)]
499 public UInt64LinksTransactionsLayer(ILinks<ulong> links, string logAddress)
500     : base(links)
501 {
502     if (string.IsNullOrEmpty(logAddress))
503     {
504         throw new ArgumentNullException(nameof(logAddress));
505     }
506     // В первой строке файла хранится последняя законченную транзакцию.
507     // При запуске это используется для проверки удачного закрытия файла лога.
508     // In the first line of the file the last committed transaction is stored.
509     // On startup, this is used to check that the log file is successfully closed.
510     var lastCommittedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
511     var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
512     if (!lastCommittedTransition.Equals(lastWrittenTransition))
513     {
514         Dispose();
515         throw new NotSupportedException("Database is damaged, autorecovery is not
516             ↳ supported yet.");
517     }
518     if (lastCommittedTransition == default)
519     {
520         FileHelpers.WriteFirst(logAddress, lastCommittedTransition);
521     }
522     _lastCommittedTransition = lastCommittedTransition;
523     // TODO: Think about a better way to calculate or store this value
524     var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
525     _lastCommittedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
526         ↳ x.TransactionId) : 0;

```

```

525     _uniqueTimestampFactory = new UniqueTimestampFactory();
526     _logAddress = logAddress;
527     _log = FileHelpers.Append(logAddress);
528     _transitions = new Queue<Transition>();
529     _transitionsPusher = new Task(TransitionsPusher);
530     _transitionsPusher.Start();
531 }
532
533 /// <summary>
534 /// <para>
535 /// Gets the link value using the specified link.
536 /// </para>
537 /// <para></para>
538 /// </summary>
539 /// <param name="link">
540 /// <para>The link.</para>
541 /// <para></para>
542 /// </param>
543 /// <returns>
544 /// <para>A list of ulong</para>
545 /// <para></para>
546 /// </returns>
547 [MethodImpl(MethodImplOptions.AggressiveInlining)]
548 public IList<ulong> GetLinkValue(ulong link) => _links.GetLink(link);
549
550 /// <summary>
551 /// <para>
552 /// Creates the substitution.
553 /// </para>
554 /// <para></para>
555 /// </summary>
556 /// <param name="substitution">
557 /// <para>The substitution.</para>
558 /// <para></para>
559 /// </param>
560 /// <returns>
561 /// <para>The created link index.</para>
562 /// <para></para>
563 /// </returns>
564 [MethodImpl(MethodImplOptions.AggressiveInlining)]
565 public override ulong Create(IList<ulong> substitution, WriteHandler<ulong> handler)
566 {
567     return _links.Create(new Link<ulong>(), (before, after) =>
568     {
569         CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
570             ↪ new Link<ulong>(before), new Link<ulong>(after)));
571         return handler(before, after);
572     });
573 }
574
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 ulong Update(IList<ulong> restriction, IList<ulong> substitution,
595     ↪ WriteHandler<ulong> handler)
596 {
597     return _links.Update(restriction, substitution, (before, after) =>
598     {
599         CommitTransition(new Transition(_uniqueTimestampFactory,
600             ↪ _currentTransactionId, new Link<ulong>(before), new Link<ulong>(after)));
601         return handler(before, after);
602     });
603 }

```

```

600     );
601 }
602
603 /// <summary>
604 /// <para>
605 /// Deletes the substitution.
606 /// </para>
607 /// <para></para>
608 /// </summary>
609 /// <param name="restriction">
610 /// <para>The substitution.</para>
611 /// <para></para>
612 /// </param>
613 [MethodImpl(MethodImplOptions.AggressiveInlining)]
614 public override ulong Delete(ICollection<ulong> restriction, WriteHandler<ulong> handler)
615 {
616     var link = restriction[_constants.IndexPart];
617     return _links.Delete(restriction, (before, after) =>
618     {
619         CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
620             ↳ before, after));
621         return handler(before, after);
622     });
623 }
624 [MethodImpl(MethodImplOptions.AggressiveInlining)]
625 private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
626     ↳ _transitions;
627 [MethodImpl(MethodImplOptions.AggressiveInlining)]
628 private void CommitTransition(Transition transition)
629 {
630     if (_currentTransaction != null)
631     {
632         Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
633     }
634     var transitions = GetCurrentTransitions();
635     transitions.Enqueue(transition);
636 }
637 [MethodImpl(MethodImplOptions.AggressiveInlining)]
638 private void RevertTransition(Transition transition)
639 {
640     if (transition.After.IsNull()) // Revert Deletion with Creation
641     {
642         _links.Create();
643     }
644     else if (transition.Before.IsNull()) // Revert Creation with Deletion
645     {
646         _links.Delete(transition.After.Index);
647     }
648     else // Revert Update
649     {
650         _links.Update(new[] { transition.After.Index, transition.Before.Source,
651             ↳ transition.Before.Target });
652     }
653 }
654 }
655 [MethodImpl(MethodImplOptions.AggressiveInlining)]
656 private void ResetCurrentTransation()
657 {
658     _currentTransactionId = 0;
659     _currentTransactionTransitions = null;
660     _currentTransaction = null;
661 }
662 [MethodImpl(MethodImplOptions.AggressiveInlining)]
663 private void PushTransitions()
664 {
665     if (_log == null || _transitions == null)
666     {
667         return;
668     }
669     for (var i = 0; i < _transitions.Count; i++)
670     {
671         var transition = _transitions.Dequeue();
672
673         _log.Write(transition);
674         _lastCommittedTransition = transition;
675     }
676 }
677 [MethodImpl(MethodImplOptions.AggressiveInlining)]
678 private void TransitionsPusher()

```

```

675     {
676         while (!Disposable.IsDisposed && _transitionsPusher != null)
677         {
678             Thread.Sleep(DefaultPushDelay);
679             PushTransitions();
680         }
681     }
682
683     /// <summary>
684     /// <para>
685     /// Begins the transaction.
686     /// </para>
687     /// <para></para>
688     /// </summary>
689     /// <returns>
690     /// <para>The transaction</para>
691     /// <para></para>
692     /// </returns>
693     [MethodImpl(MethodImplOptions.AggressiveInlining)]
694     public Transaction BeginTransaction() => new Transaction(this);
695     [MethodImpl(MethodImplOptions.AggressiveInlining)]
696     private void DisposeTransitions()
697     {
698         try
699         {
700             var pusher = _transitionsPusher;
701             if (pusher != null)
702             {
703                 _transitionsPusher = null;
704                 pusher.Wait();
705             }
706             if (_transitions != null)
707             {
708                 PushTransitions();
709             }
710             _log.DisposeIfPossible();
711             FileHelpers.WriteFirst(_logAddress, _lastCommittedTransition);
712         }
713         catch (Exception ex)
714         {
715             ex.Ignore();
716         }
717     }
718
719     #region DisposalBase
720
721     /// <summary>
722     /// <para>
723     /// Disposes the manual.
724     /// </para>
725     /// <para></para>
726     /// </summary>
727     /// <param name="manual">
728     /// <para>The manual.</para>
729     /// <para></para>
730     /// </param>
731     /// <param name="wasDisposed">
732     /// <para>The was disposed.</para>
733     /// <para></para>
734     /// </param>
735     [MethodImpl(MethodImplOptions.AggressiveInlining)]
736     protected override void Dispose(bool manual, bool wasDisposed)
737     {
738         if (!wasDisposed)
739         {
740             DisposeTransitions();
741         }
742         base.Dispose(manual, wasDisposed);
743     }
744
745     #endregion
746 }
747 }

```

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

```

1 using System;
2 using System.IO;
3 using Platform.Data.Doublets.Decorators;
4 using Xunit;

```

```

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

```

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

```

1 using System.IO;
2 using Platform.Data.Doublets.Memory.United.Generic;
3 using Platform.Memory;
4 using Xunit;
5
6
7 namespace Platform.Data.Doublets.Tests
8 {
9     public static class ILinksBasicTests
10     {
11         [Fact]
12         public static void DeleteAllUsages()
13         {
14             var mem = new HeapResizableDirectMemory();
15             var links = new UnitedMemoryLinks<uint>(mem);
16
17             var root = links.CreatePoint();
18
19             var a = links.CreatePoint();
20             var b = links.CreatePoint();

```

```

21         links.CreateAndUpdate(a, root);
22         links.CreateAndUpdate(b, root);
23
24         Assert.Equal(5U, links.Count());
25
26         links.DeleteAllUsages(root);
27
28         Assert.Equal(3U, links.Count());
29     }
30
31     [Fact]
32     public static void FfiDeleteAllUsages()
33     {
34         File.Delete("db.links");
35         var links = new FFI.UnitedMemoryLinks<uint>("db.links");
36
37         var root = links.CreatePoint();
38
39         var a = links.CreatePoint();
40         var b = links.CreatePoint();
41
42         links.CreateAndUpdate(a, root);
43         links.CreateAndUpdate(b, root);
44
45         Assert.Equal(5U, links.Count());
46
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         }
23     }

```

```

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

```

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

```

1  using Xunit;
2  using Platform.Scopes;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Reflection;
6  using Platform.Data.Doublets.Memory.United.Generic;
7  using Platform.Data.Doublets.Memory.United.Specific;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class ScopeTests
12     {
13         [Fact]
14         public static void SingleDependencyTest()
15         {
16             using (var scope = new Scope())
17             {
18                 scope.IncludeAssemblyOf<IMemory>();
19                 var instance = scope.Use<IDirectMemory>();
20                 Assert.IsType<HeapResizableDirectMemory>(instance);
21             }
22         }
23
24         [Fact]
25         public static void CascadeDependencyTest()
26         {
27             using (var scope = new Scope())
28             {
29                 scope.Include<TemporaryFileMappedResizableDirectMemory>();
30                 scope.Include<UInt64UnitedMemoryLinks>();

```

```

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

```

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

```

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

```



```

44     }
45 }
46 private static void
    ↳ UsingWithExternalReferences<TLinkAddress>(Action<ILinks<TLinkAddress>> action) where
    ↳ TLinkAddress : struct
47 {
48     var constants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
    ↳ true);
49     using (var dataMemory = new HeapResizableDirectMemory())
50     using (var indexMemory = new HeapResizableDirectMemory())
51     using (var memory = new SplitMemoryLinks<TLinkAddress>(dataMemory, indexMemory,
    ↳ SplitMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, constants))
52     {
53         action(memory);
54     }
55 }
56 }
57 }

```

### 1.125 ./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:
    ↳ true);
40             using (var dataMemory = new HeapResizableDirectMemory())
41             using (var indexMemory = new HeapResizableDirectMemory())
42             using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory,
    ↳ UInt32SplitMemoryLinks.DefaultLinksSizeStep, constants))
43             {
44                 action(memory);
45             }
46         }
47     }
48 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLinkAddress = System.UInt64;

```

```

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

```

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

```

1 using System.Collections.Generic;
2 using Xunit;
3 using Platform.Ranges;
4 using Platform.Numbers;
5 using Platform.Random;
6 using Platform.Setters;
7 using Platform.Converters;
8
9 namespace Platform.Data.Doublets.Tests
10 {
11     public static class TestExtensions
12     {
13         public static void TestCRUDOperations<T>(this ILinks<T> links)
14         {
15             var constants = links.Constants;
16
17             var equalityComparer = EqualityComparer<T>.Default;
18
19             var zero = default(T);
20             var one = Arithmetic.Increment(zero);
21
22             // Create Link
23             Assert.True(equalityComparer.Equals(links.Count(), zero));
24
25             var setter = new Setter<T>(constants.Null);
26             links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
27
28             Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
29
30             var linkAddress = links.Create();
31

```

```

32     var link = new Link<T>(links.GetLink(linkAddress));
33
34     Assert.True(link.Count == 3);
35     Assert.True(equalityComparer.Equals(link.Index, linkAddress));
36     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
37     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
38
39     Assert.True(equalityComparer.Equals(links.Count(), one));
40
41     // Get first link
42     setter = new Setter<T>(constants.Null);
43     links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
44
45     Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
46
47     // Update link to reference itself
48     links.Update(linkAddress, linkAddress, linkAddress);
49
50     link = new Link<T>(links.GetLink(linkAddress));
51
52     Assert.True(equalityComparer.Equals(link.Source, linkAddress));
53     Assert.True(equalityComparer.Equals(link.Target, linkAddress));
54
55     // Update link to reference null (prepare for delete)
56     var updated = links.Update(linkAddress, constants.Null, constants.Null);
57
58     Assert.True(equalityComparer.Equals(updated, linkAddress));
59
60     link = new Link<T>(links.GetLink(linkAddress));
61
62     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
63     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
64
65     // Delete link
66     links.Delete(linkAddress);
67
68     Assert.True(equalityComparer.Equals(links.Count(), zero));
69
70     setter = new Setter<T>(constants.Null);
71     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
72
73     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
74 }
75
76 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
77 {
78     // Constants
79     var constants = links.Constants;
80     var equalityComparer = EqualityComparer<T>.Default;
81
82     var zero = default(T);
83     var one = Arithmetic.Increment(zero);
84     var two = Arithmetic.Increment(one);
85
86     var h106E = new Hybrid<T>(106L, isExternal: true);
87     var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
88     var h108E = new Hybrid<T>(-108L);
89
90     Assert.Equal(106L, h106E.AbsoluteValue);
91     Assert.Equal(107L, h107E.AbsoluteValue);
92     Assert.Equal(108L, h108E.AbsoluteValue);
93
94     // Create Link (External -> External)
95     var linkAddress1 = links.Create();
96
97     links.Update(linkAddress1, h106E, h108E);
98
99     var link1 = new Link<T>(links.GetLink(linkAddress1));
100
101     Assert.True(equalityComparer.Equals(link1.Source, h106E));
102     Assert.True(equalityComparer.Equals(link1.Target, h108E));
103
104     // Create Link (Internal -> External)
105     var linkAddress2 = links.Create();
106
107     links.Update(linkAddress2, linkAddress1, h108E);
108
109     var link2 = new Link<T>(links.GetLink(linkAddress2));
110
111     Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));

```

```

112 Assert.True(equalityComparer.Equals(link2.Target, h108E));
113
114 // Create Link (Internal -> Internal)
115 var linkAddress3 = links.Create();
116
117 links.Update(linkAddress3, linkAddress1, linkAddress2);
118
119 var link3 = new Link<T>(links.GetLink(linkAddress3));
120
121 Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
122 Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
123
124 // Search for created link
125 var setter1 = new Setter<T>(constants.Null);
126 links.Each(h106E, h108E, setter1.SetAndReturnFalse);
127
128 Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
129
130 // Search for nonexistent link
131 var setter2 = new Setter<T>(constants.Null);
132 links.Each(h106E, h107E, setter2.SetAndReturnFalse);
133
134 Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
135
136 // Update link to reference null (prepare for delete)
137 var updated = links.Update(linkAddress3, constants.Null, constants.Null);
138
139 Assert.True(equalityComparer.Equals(updated, linkAddress3));
140
141 link3 = new Link<T>(links.GetLink(linkAddress3));
142
143 Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
144 Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
145
146 // Delete link
147 links.Delete(linkAddress3);
148
149 Assert.True(equalityComparer.Equals(links.Count(), two));
150
151 var setter3 = new Setter<T>(constants.Null);
152 links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
153
154 Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
155 }
156
157 public static void TestMultipleCreationsAndDeletions<TLinkAddress>(this
    ↳ ILinks<TLinkAddress> links, int numberOfOperations)
158 {
159     for (int i = 0; i < numberOfOperations; i++)
160     {
161         links.Create();
162     }
163     for (int i = 0; i < numberOfOperations; i++)
164     {
165         links.Delete(links.Count());
166     }
167 }
168
169 public static void TestMultipleRandomCreationsAndDeletions<TLinkAddress>(this
    ↳ ILinks<TLinkAddress> links, int maximumOperationsPerCycle)
170 {
171     var comparer = Comparer<TLinkAddress>.Default;
172     var addressToUInt64Converter = CheckedConverter<TLinkAddress, ulong>.Default;
173     var uint64ToAddressConverter = CheckedConverter<ulong, TLinkAddress>.Default;
174     for (var N = 1; N < maximumOperationsPerCycle; N++)
175     {
176         var random = new System.Random(N);
177         var created = 0UL;
178         var deleted = 0UL;
179         for (var i = 0; i < N; i++)
180         {
181             var linksCount = addressToUInt64Converter.Convert(links.Count());
182             var createPoint = random.NextBoolean();
183             if (linksCount >= 2 && createPoint)
184             {
185                 var linksAddressRange = new Range<ulong>(1, linksCount);
186                 TLinkAddress source = uint64ToAddressConverter.Convert(random.NextUInt64
    ↳ (linksAddressRange));

```

```

187         TLinkAddress target = UInt64ToAddressConverter.Convert(random.NextUInt64_
        ↪ (linksAddressRange));
        ↪ //-V3086
188     var resultLink = links.GetOrCreate(source, target);
189     if (comparer.Compare(resultLink,
        ↪ UInt64ToAddressConverter.Convert(linksCount)) > 0)
190     {
191         created++;
192     }
193 }
194 else
195 {
196     links.Create();
197     created++;
198 }
199 }
200 Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
201 for (var i = 0; i < N; i++)
202 {
203     TLinkAddress link = UInt64ToAddressConverter.Convert((ulong)i + 1UL);
204     if (links.Exists(link))
205     {
206         links.Delete(link);
207         deleted++;
208     }
209 }
210 Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
211 }
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.Numbers.Raw;
4  using Platform.Memory;
5  using Platform.Numbers;
6  using Xunit;
7  using Xunit.Abstractions;
8  using TLinkAddress = System.UInt64;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     public class UInt64LinksExtensionsTests
13     {
14         public static ILinks<TLinkAddress> CreateLinks() => CreateLinks<TLinkAddress>(new
        ↪ Platform.IO.TemporaryFile());
15
16         public static ILinks<TLinkAddress> CreateLinks<TLinkAddress>(string dataDBFilename)
        ↪ where TLinkAddress : struct
17         {
18             var linksConstants = new
        ↪ LinksConstants<TLinkAddress>(enableExternalReferencesSupport: true);
19             return new UnitedMemoryLinks<TLinkAddress>(new
        ↪ FileMappedResizableDirectMemory(dataDBFilename),
        ↪ UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
        ↪ IndexTreeType.Default);
20         }
21         [Fact]
22         public void FormatStructureWithExternalReferenceTest()
23         {
24             ILinks<TLinkAddress> links = CreateLinks();
25             TLinkAddress zero = default;
26             var one = Arithmetic.Increment(zero);
27             var markerIndex = one;
28             var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
29             var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
        ↪ markerIndex));
30             AddressToRawNumberConverter<TLinkAddress> addressToNumberConverter = new();
31             var numberAddress = addressToNumberConverter.Convert(1);
32             var numberLink = links.GetOrCreate(numberMarker, numberAddress);
33             var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
        ↪ true);
34             Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
35         }
36     }
37 }

```

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

```

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

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLinkAddress = System.UInt64;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public unsafe static class UnitedMemoryUInt64LinksTests
12     {
13         [Fact]
14         public static void CRUDTest()
15         {
16             Using(links => links.TestCRUDOperations());
17         }
18
19         [Fact]
20         public static void RawNumbersCRUDTest()
21         {
22             Using(links => links.TestRawNumbersCRUDOperations());
23         }
24
25         [Fact]
26         public static void MultipleRandomCreationsAndDeletionsTest()
27         {
28             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().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>>());

```

35

36

37

38

}

}

}

}

## Index

`./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.cs`, 460  
`./csharp/Platform.Data.Doublets.Tests/ILinksBasicTests.cs`, 461  
`./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs`, 462  
`./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs`, 462  
`./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs`, 463  
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs`, 464  
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs`, 465  
`./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`, 26  
`./csharp/Platform.Data.Doublets/DoubletComparer.cs`, 28  
`./csharp/Platform.Data.Doublets/FFI/UInt32UnitedMemoryLinks.cs`, 29  
`./csharp/Platform.Data.Doublets/FFI/UnitedMemoryLinks.cs`, 30  
`./csharp/Platform.Data.Doublets/ILinks.cs`, 40  
`./csharp/Platform.Data.Doublets/ILinksExtensions.cs`, 40  
`./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs`, 60  
`./csharp/Platform.Data.Doublets/Link.cs`, 61  
`./csharp/Platform.Data.Doublets/LinkExtensions.cs`, 68  
`./csharp/Platform.Data.Doublets/LinksOperatorBase.cs`, 69  
`./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs`, 69  
`./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs`, 70  
`./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs`, 71  
`./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs`, 72  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 74  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs`, 81  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 88  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 92  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 96  
`./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`, 115  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 120  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs`, 123  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 127  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs`, 131  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs`, 134  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs`, 138  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs`, 156  
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs`, 159  
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs`, 160  
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 162  
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase.cs`, 168  
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 174  
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 178



./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 182  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs, 186  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 190  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 196  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 202  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 203  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 206  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 210  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 214  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 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, 249  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 253  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.cs, 258  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 264  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 265  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 269  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 272  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 276  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 280  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 286  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 287  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 297  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 303  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 310  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 315  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 319  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 322  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 328  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 332  
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs, 335  
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs, 338  
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs, 351  
./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs, 354  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 356  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 362  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 367  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs, 371  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 375  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 378  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 382  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 388  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 389  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 396  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 401  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 407  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 412  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 416  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 420  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 425  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 429  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 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, 444

./csharp/Platform.Data.Doublets/SynchronizedLinks.cs, 444  
./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs, 447  
./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 450