

## LinksPlatform's Platform.Data.Doublets Class Library

### 1.1 ./csharp/Platform.Data.Doublets/CriterionMatchers/TargetMatcher.cs

```
1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Interfaces;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.CriterionMatchers
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the target matcher.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="LinksOperatorBase{TLink}"/>
16    /// <seealso cref="ICriterionMatcher{TLink}"/>
17    public class TargetMatcher<TLink> : LinksOperatorBase<TLink>, ICriterionMatcher<TLink>
18    {
19        private static readonly EqualityComparer<TLink> _equalityComparer =
20            ↪ EqualityComparer<TLink>.Default;
21        private readonly TLink _targetToMatch;
22
23        /// <summary>
24        /// <para>
25        /// Initializes a new <see cref="TargetMatcher"/> instance.
26        /// </para>
27        /// <para></para>
28        /// </summary>
29        /// <param name="links">
30        /// <para>A links.</para>
31        /// </param>
32        /// <param name="targetToMatch">
33        /// <para>A target to match.</para>
34        /// </param>
35        /// </summary>
36        [MethodImpl(MethodImplOptions.AggressiveInlining)]
37        public TargetMatcher(ILinks<TLink> links, TLink targetToMatch) : base(links) =>
38            ↪ _targetToMatch = targetToMatch;
39
40        /// <summary>
41        /// <para>
42        /// Determines whether this instance is matched.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="link">
47        /// <para>The link.</para>
48        /// </param>
49        /// <returns>
50        /// <para>The bool</para>
51        /// </returns>
52        [MethodImpl(MethodImplOptions.AggressiveInlining)]
53        public bool IsMatched(TLink link) => _equalityComparer.Equals(_links.GetTarget(link),
54            ↪ _targetToMatch);
55    }
56 }
```

### 1.2 ./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUniquenessAndUsagesResolver.cs

```
1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Delegates;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Decorators
9 {
10    /// <summary>
11    /// <para>
12    /// Represents the links cascade uniqueness and usages resolver.
13    /// </para>
14    /// <para></para>
15    /// </summary>
16    /// <seealso cref="LinksUniquenessResolver{TLink}"/>
```

```

17 public class LinksCascadeUniquenessAndUsagesResolver<TLink> : LinksUniquenessResolver<TLink>
18 {
19     /// <summary>
20     /// <para>
21     /// Initializes a new <see cref="LinksCascadeUniquenessAndUsagesResolver"/> instance.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     /// <param name="links">
26     /// <para>A links.</para>
27     /// <para></para>
28     /// </param>
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     public LinksCascadeUniquenessAndUsagesResolver(ILinks<TLink> links) : base(links) { }
31
32     /// <summary>
33     /// <para>
34     /// Resolves the address change conflict using the specified old link address.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="oldLinkAddress">
39     /// <para>The old link address.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="newLinkAddress">
43     /// <para>The new link address.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The link</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
52     ↪ newLinkAddress, WriteHandler<TLink>? handler)
53     {
54         var constants = _links.Constants;
55         WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
56         ↪ handler);
57         // Use Facade (the last decorator) to ensure recursion working correctly
58         handlerState.Apply(_facade.MergeUsages(oldLinkAddress, newLinkAddress,
59         ↪ handlerState.Handler));
60         handlerState.Apply(base.ResolveAddressChangeConflict(oldLinkAddress, newLinkAddress,
61         ↪ handlerState.Handler));
62         return handlerState.Result;
63     }
64 }
65 }

```

### 1.3 ./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUsagesResolver.cs

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Delegates;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.Data.Doublets.Decorators
9 {
10     /// <remarks>
11     /// <para>Must be used in conjunction with NonNullContentsLinkDeletionResolver.</para>
12     /// <para>Должен использоваться вместе с NonNullContentsLinkDeletionResolver.</para>
13     /// </remarks>
14     public class LinksCascadeUsagesResolver<TLink> : LinksDecoratorBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="LinksCascadeUsagesResolver"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="links">
23         /// <para>A links.</para>
24         /// <para></para>
25         /// </param>
26         [MethodImpl(MethodImplOptions.AggressiveInlining)]
27         public LinksCascadeUsagesResolver(ILinks<TLink> links) : base(links) { }

```

```

28
29     /// <summary>
30     /// <para>
31     /// Deletes the restriction.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     /// <param name="restriction">
36     /// <para>The restriction.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public override TLink Delete(ICollection<TLink>? restriction, WriteHandler<TLink>? handler)
41     {
42         var constants = _links.Constants;
43         WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
44             ↪ handler);
45         var linkIndex = restriction[_constants.IndexPart];
46         // Use Facade (the last decorator) to ensure recursion working correctly
47         handlerState.Apply(_facade.DeleteAllUsages(linkIndex, handlerState.Handler));
48         handlerState.Apply(_links.Delete(restriction, handlerState.Handler));
49         return handlerState.Result;
50     }
51 }

```

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

```

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

```

```

53     /// </summary>
54     public ILinks<TLink> Facade
55     {
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         get => _facade;
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         set
60         {
61             _facade = value;
62             if (_links is LinksDecoratorBase<TLink> decorator)
63             {
64                 decorator.Facade = value;
65             }
66         }
67     }
68
69     /// <summary>
70     /// <para>
71     /// Initializes a new <see cref="LinksDecoratorBase"/> instance.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <param name="links">
76     /// <para>A links.</para>
77     /// <para></para>
78     /// </param>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     protected LinksDecoratorBase(ILinks<TLink> links) : base(links)
81     {
82         _constants = links.Constants;
83         Facade = this;
84     }
85
86     /// <summary>
87     /// <para>
88     /// Counts the restriction.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="restriction">
93     /// <para>The restriction.</para>
94     /// <para></para>
95     /// </param>
96     /// <returns>
97     /// <para>The link</para>
98     /// <para></para>
99     /// </returns>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    public virtual TLink Count(IList<TLink>? restriction) => _links.Count(restriction);
102
103    /// <summary>
104    /// <para>
105    /// Eaches the handler.
106    /// </para>
107    /// <para></para>
108    /// </summary>
109    /// <param name="handler">
110    /// <para>The handler.</para>
111    /// <para></para>
112    /// </param>
113    /// <param name="restriction">
114    /// <para>The restriction.</para>
115    /// <para></para>
116    /// </param>
117    /// <returns>
118    /// <para>The link</para>
119    /// <para></para>
120    /// </returns>
121    [MethodImpl(MethodImplOptions.AggressiveInlining)]
122    public virtual TLink Each(IList<TLink>? restriction, ReadHandler<TLink>? handler) =>
123        ↪ _links.Each(restriction, handler);
124
125    /// <summary>
126    /// <para>
127    /// Creates the restriction.
128    /// </para>
129    /// <para></para>
130    /// </summary>

```

```

130     /// <param name="restriction">
131     /// <para>The restriction.</para>
132     /// <para></para>
133     /// </param>
134     /// <returns>
135     /// <para>The link</para>
136     /// <para></para>
137     /// </returns>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     public virtual TLink Create(IList<TLink>? substitution, WriteHandler<TLink>? handler) =>
140         ↪ _links.Create(substitution, handler);
141
142     /// <summary>
143     /// <para>
144     /// Updates the restriction.
145     /// </para>
146     /// <para></para>
147     /// </summary>
148     /// <param name="restriction">
149     /// <para>The restriction.</para>
150     /// <para></para>
151     /// </param>
152     /// <param name="substitution">
153     /// <para>The substitution.</para>
154     /// <para></para>
155     /// </param>
156     /// <returns>
157     /// <para>The link</para>
158     /// <para></para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public virtual TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
162         ↪ WriteHandler<TLink>? handler) => _links.Update(restriction, substitution, handler);
163
164     /// <summary>
165     /// <para>
166     /// Deletes the restriction.
167     /// </para>
168     /// <para></para>
169     /// </summary>
170     /// <param name="restriction">
171     /// <para>The restriction.</para>
172     /// <para></para>
173     /// </param>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     public virtual TLink Delete(IList<TLink>? restriction, WriteHandler<TLink>? handler) =>
176         ↪ _links.Delete(restriction, handler);
177 }
178 }

```

## 1.5 ./csharp/Platform.Data.Doublets/Decorators/LinksDisposableDecoratorBase.cs

```

1 using System.Runtime.CompilerServices;
2 using Platform.Disposables;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5 #pragma warning disable CA1063 // Implement IDisposable Correctly
6
7 namespace Platform.Data.Doublets.Decorators
8 {
9     /// <summary>
10     /// <para>
11     /// Represents the links disposable decorator base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksDecoratorBase{TLink}"/>
16     /// <seealso cref="ILinks{TLink}"/>
17     /// <seealso cref="System.IDisposable"/>
18     public abstract class LinksDisposableDecoratorBase<TLink> : LinksDecoratorBase<TLink>,
19         ↪ ILinks<TLink>, System.IDisposable
20     {
21         /// <summary>
22         /// <para>
23         /// Represents the disposable with multiple calls allowed.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <seealso cref="Disposable"/>

```

```

27 protected class DisposableWithMultipleCallsAllowed : Disposable
28 {
29     /// <summary>
30     /// <para>
31     /// Initializes a new <see cref="DisposableWithMultipleCallsAllowed"/> instance.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     /// <param name="disposal">
36     /// <para>A disposal.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public DisposableWithMultipleCallsAllowed(Disposal disposal) : base(disposal) { }
41
42     /// <summary>
43     /// <para>
44     /// Gets the allow multiple dispose calls value.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     protected override bool AllowMultipleDisposeCalls
49     {
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         get => true;
52     }
53 }
54
55 /// <summary>
56 /// <para>
57 /// The disposable.
58 /// </para>
59 /// <para></para>
60 /// </summary>
61 protected readonly DisposableWithMultipleCallsAllowed Disposable;
62
63 /// <summary>
64 /// <para>
65 /// Initializes a new <see cref="LinksDisposableDecoratorBase"/> instance.
66 /// </para>
67 /// <para></para>
68 /// </summary>
69 /// <param name="links">
70 /// <para>A links.</para>
71 /// <para></para>
72 /// </param>
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 protected LinksDisposableDecoratorBase(ILinks<TLink> links) : base(links) => Disposable
75     ↳ = new DisposableWithMultipleCallsAllowed(Dispose);
76
77 [MethodImpl(MethodImplOptions.AggressiveInlining)]
78 ~LinksDisposableDecoratorBase() => Disposable.Destruct();
79
80 /// <summary>
81 /// <para>
82 /// Disposes this instance.
83 /// </para>
84 /// <para></para>
85 /// </summary>
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 public void Dispose() => Disposable.Dispose();
88
89 /// <summary>
90 /// <para>
91 /// Disposes the manual.
92 /// </para>
93 /// <para></para>
94 /// </summary>
95 /// <param name="manual">
96 /// <para>The manual.</para>
97 /// <para></para>
98 /// </param>
99 /// <param name="wasDisposed">
100 /// <para>The was disposed.</para>
101 /// <para></para>
102 /// </param>
103 [MethodImpl(MethodImplOptions.AggressiveInlining)]
104 protected virtual void Dispose(bool manual, bool wasDisposed)

```

```

104     {
105         if (!wasDisposed)
106         {
107             _links.DisposeIfPossible();
108         }
109     }
110 }
111 }

```

## 1.6 ./csharp/Platform.Data.Doublets/Decorators/LinksInnerReferenceExistenceValidator.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     // TODO: Make LinksExternalReferenceValidator. A layer that checks each link to exist or to
11     // ↳ be external (hybrid link's raw number).
12     /// <summary>
13     /// <para>
14     /// Represents the links inner reference existence validator.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="LinksDecoratorBase{TLink}" />
19     public class LinksInnerReferenceExistenceValidator<TLink> : LinksDecoratorBase<TLink>
20     {
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="LinksInnerReferenceExistenceValidator" /> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public LinksInnerReferenceExistenceValidator(ILinks<TLink> links) : base(links) { }
33
34         /// <summary>
35         /// <para>
36         /// Eaches the handler.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="handler">
41         /// <para>The handler.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="restriction">
45         /// <para>The restriction.</para>
46         /// <para></para>
47         /// </param>
48         /// <returns>
49         /// <para>The link</para>
50         /// <para></para>
51         /// </returns>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         public override TLink Each(IList<TLink>? restriction, ReadHandler<TLink>? handler)
54         {
55             var links = _links;
56             links.EnsureInnerReferenceExists(restriction, nameof(restriction));
57             return links.Each(restriction, handler);
58         }
59
60         /// <summary>
61         /// <para>
62         /// Updates the restriction.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <param name="restriction">
67         /// <para>The restriction.</para>
68         /// <para></para>
69         /// </param>

```

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links itself constant to self reference resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase<TLink>" />
17     public class LinksItselfConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
18     {
19         private static readonly EqualityComparer<TLink> _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="LinksItselfConstantToSelfReferenceResolver" /> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         public LinksItselfConstantToSelfReferenceResolver(ICollection<TLink> links) : base(links) { }
33
34         /// <summary>
35         /// <para>
36         /// Eaches the handler.
37         /// </para>

```



```

38     /// <para></para>
39     /// </summary>
40     /// <param name="handler">
41     /// <para>The handler.</para>
42     /// <para></para>
43     /// </param>
44     /// <param name="restriction">
45     /// <para>The restriction.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public override TLink Each(ICollection<TLink>? restriction, ReadHandler<TLink>? handler)
54     {
55         var constants = _constants;
56         var itselfConstant = constants.Itself;
57         if (!_equalityComparer.Equals(constants.Any, itselfConstant) &&
58             ↪ restriction.Contains(itselfConstant))
59         {
60             // Itself constant is not supported for Each method right now, skipping execution
61             return constants.Continue;
62         }
63         return _links.Each(restriction, handler);
64     }
65     /// <summary>
66     /// <para>
67     /// Updates the restriction.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <param name="restriction">
72     /// <para>The restriction.</para>
73     /// <para></para>
74     /// </param>
75     /// <param name="substitution">
76     /// <para>The substitution.</para>
77     /// <para></para>
78     /// </param>
79     /// <returns>
80     /// <para>The link</para>
81     /// <para></para>
82     /// </returns>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     public override TLink Update(ICollection<TLink>? restriction, ICollection<TLink>? substitution,
85         ↪ WriteHandler<TLink>? handler) => _links.Update(restriction,
86         ↪ _links.ResolveConstantAsSelfReference(_constants.Itself, restriction, substitution),
87         ↪ handler);
88     }
89 }

```

## 1.8 ./csharp/Platform.Data.Doublets/Decorators/LinksNonExistentDependenciesCreator.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <remarks>
11     /// Not practical if newSource and newTarget are too big.
12     /// To be able to use practical version we should allow to create link at any specific
13     ↪ location inside ResizableDirectMemoryLinks.
14     /// This in turn will require to implement not a list of empty links, but a list of ranges
15     ↪ to store it more efficiently.
16     /// </remarks>
17     public class LinksNonExistentDependenciesCreator<TLink> : LinksDecoratorBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="LinksNonExistentDependenciesCreator"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>

```

```

23     /// <param name="links">
24     /// <para>A links.</para>
25     /// <para></para>
26     /// </param>
27     [MethodImpl(MethodImplOptions.AggressiveInlining)]
28     public LinksNonExistentDependenciesCreator(ILinks<TLink> links) : base(links) { }
29
30     /// <summary>
31     /// <para>
32     /// Updates the restriction.
33     /// </para>
34     /// <para></para>
35     /// </summary>
36     /// <param name="restriction">
37     /// <para>The restriction.</para>
38     /// <para></para>
39     /// </param>
40     /// <param name="substitution">
41     /// <para>The substitution.</para>
42     /// <para></para>
43     /// </param>
44     /// <returns>
45     /// <para>The link</para>
46     /// <para></para>
47     /// </returns>
48     [MethodImpl(MethodImplOptions.AggressiveInlining)]
49     public override TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
50     ↪ WriteHandler<TLink>? handler)
51     {
52         var constants = _constants;
53         var links = _links;
54         links.EnsureCreated(substitution[constants.SourcePart],
55         ↪ substitution[constants.TargetPart]);
56         return links.Update(restriction, substitution, handler);
57     }
58 }

```

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

```

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

```

```

41     /// </param>
42     /// <returns>
43     /// <para>The link</para>
44     /// <para></para>
45     /// </returns>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public override TLink Create(ICollection<TLink>? substitution, WriteHandler<TLink>? handler)
48     {
49         return _links.CreatePoint(handler);
50     }
51
52     /// <summary>
53     /// <para>
54     /// Updates the substitution.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     /// <param name="restriction">
59     /// <para>The substitution.</para>
60     /// <para></para>
61     /// </param>
62     /// <param name="substitution">
63     /// <para>The substitution.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public override TLink Update(ICollection<TLink>? restriction, ICollection<TLink>? substitution,
72         WriteHandler<TLink>? handler) => _links.Update(restriction,
73         _links.ResolveConstantAsSelfReference(_constants.Null, restriction, substitution),
74         handler);
75 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links uniqueness resolver.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLink}" />
17     public class LinksUniquenessResolver<TLink> : LinksDecoratorBase<TLink>
18     {
19         private static readonly EqualityComparer<TLink> _equalityComparer =
20             EqualityComparer<TLink>.Default;
21
22         /// <summary>
23         /// <para>
24         /// Initializes a new <see cref="LinksUniquenessResolver" /> instance.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         [MethodImpl(MethodImplOptions.AggressiveInlining)]
33         public LinksUniquenessResolver(ICollection<TLink> links) : base(links) { }
34
35         /// <summary>
36         /// <para>
37         /// Updates the restriction.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         /// <param name="restriction">

```

```

41     /// <para>The restriction.</para>
42     /// <para></para>
43     /// </param>
44     /// <param name="substitution">
45     /// <para>The substitution.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public override TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
54     ↪ WriteHandler<TLink>? handler)
55     {
56         var constants = _constants;
57         var links = _links;
58         var newLinkAddress = links.SearchOrDefault(substitution[constants.SourcePart],
59     ↪ substitution[constants.TargetPart]);
60         if (_equalityComparer.Equals(newLinkAddress, default))
61         {
62             return links.Update(restriction, substitution, handler);
63         }
64         return ResolveAddressChangeConflict(restriction[constants.IndexPart],
65     ↪ newLinkAddress, handler);
66     }
67
68     /// <summary>
69     /// <para>
70     /// Resolves the address change conflict using the specified old link address.
71     /// </para>
72     /// <para></para>
73     /// </summary>
74     /// <param name="oldLinkAddress">
75     /// <para>The old link address.</para>
76     /// <para></para>
77     /// </param>
78     /// <param name="newLinkAddress">
79     /// <para>The new link address.</para>
80     /// <para></para>
81     /// </param>
82     /// <returns>
83     /// <para>The new link address.</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected virtual TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
88     ↪ newLinkAddress, WriteHandler<TLink>? handler)
89     {
90         if (!_equalityComparer.Equals(oldLinkAddress, newLinkAddress) &&
91     ↪ _links.Exists(oldLinkAddress))
92         {
93             return _facade.Delete(oldLinkAddress, handler);
94         }
95         return _links.Constants.Continue;
96     }
97 }
98
99 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Decorators
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the links uniqueness validator.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksDecoratorBase{TLink}">
17     public class LinksUniquenessValidator<TLink> : LinksDecoratorBase<TLink>
18     {

```

```

19     /// <summary>
20     /// <para>
21     /// Initializes a new <see cref="LinksUniquenessValidator"/> instance.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     /// <param name="links">
26     /// <para>A links.</para>
27     /// <para></para>
28     /// </param>
29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     public LinksUniquenessValidator(ILinks<TLink> links) : base(links) { }
31
32     /// <summary>
33     /// <para>
34     /// Updates the restriction.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="restriction">
39     /// <para>The restriction.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="substitution">
43     /// <para>The substitution.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The link</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public override TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
52     ↪ WriteHandler<TLink>? handler)
53     {
54         var links = _links;
55         var constants = _constants;
56         links.EnsureDoesNotExists(substitution[constants.SourcePart],
57         ↪ substitution[constants.TargetPart]);
58         return links.Update(restriction, substitution, handler);
59     }

```

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

```

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

```

```

35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="restriction">
39     /// <para>The restriction.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="substitution">
43     /// <para>The substitution.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The link</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public override TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
    ↪ WriteHandler<TLink>? handler)
52     {
53         var links = _links;
54         links.EnsureNoUsages(restriction[_constants.IndexPart]);
55         return links.Update(restriction, substitution, handler);
56     }
57
58     /// <summary>
59     /// <para>
60     /// Deletes the restriction.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="restriction">
65     /// <para>The restriction.</para>
66     /// <para></para>
67     /// </param>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     public override TLink Delete(IList<TLink>? restriction, WriteHandler<TLink>? handler)
70     {
71         var link = restriction[_constants.IndexPart];
72         var links = _links;
73         links.EnsureNoUsages(link);
74         return links.Delete(restriction, handler);
75     }
76 }
77 }

```

### 1.13 ./csharp/Platform.Data.Doublets/Decorators/LoggingDecorator.cs

```

1  using System.Collections.Generic;
2  using System.IO;
3  using Platform.Delegates;
4
5  namespace Platform.Data.Doublets.Decorators
6  {
7      public class LoggingDecorator<TLink> : LinksDecoratorBase<TLink>
8      {
9          private readonly Stream _logStream;
10         private readonly StreamWriter _logStreamWriter;
11         public LoggingDecorator(ILinks<TLink> links, Stream logStream) : base(links)
12         {
13             _logStream = logStream;
14             _logStreamWriter = new StreamWriter(_logStream);
15             _logStreamWriter.AutoFlush = true;
16         }
17
18         public override TLink Create(IList<TLink>? substitution, WriteHandler<TLink>? handler)
19         {
20             WriteHandlerState<TLink> handlerState = new(_constants.Continue, _constants.Break,
    ↪ handler);
21             return base.Create(substitution, (before, after) =>
22             {
23                 if (handlerState.Handler != null)
24                 {
25                     handlerState.Apply(handlerState.Handler(before, after));
26                 }
27                 _logStreamWriter.WriteLine($"Create. Before: {new Link<TLink>(before)}. After:
    ↪ {new Link<TLink>(after)}");
28                 return _constants.Continue;
29             });
30         }
31     }

```

```

32     public override TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
    ↪ WriteHandler<TLink>? handler)
33     {
34         WriteHandlerState<TLink> handlerState = new(_constants.Continue, _constants.Break,
    ↪ handler);
35         return base.Update(restriction, substitution, (before, after) =>
36         {
37             if (handlerState.Handler != null)
38             {
39                 handlerState.Apply(handlerState.Handler(before, after));
40             }
41             _logStreamWriter.WriteLine($"Update. Before: {new Link<TLink>(before)}. After:
    ↪ {new Link<TLink>(after)}");
42             return _constants.Continue;
43         });
44     }
45
46     public override TLink Delete(IList<TLink>? restriction, WriteHandler<TLink>? handler)
47     {
48         WriteHandlerState<TLink> handlerState = new(_constants.Continue, _constants.Break,
    ↪ handler);
49         return base.Delete(restriction, (before, after) =>
50         {
51             if (handlerState.Handler != null)
52             {
53                 handlerState.Apply(handlerState.Handler(before, after));
54             }
55             _logStreamWriter.WriteLine($"Delete. Before: {new Link<TLink>(before)}. After:
    ↪ {new Link<TLink>(after)}");
56             return _constants.Continue;
57         });
58     }
59 }
60 }

```

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

```

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

```

```

43     public override TLink Delete(ICollection<TLink>? restriction, WriteHandler<TLink>? handler)
44     {
45         var linkIndex = restriction[_constants.IndexPart];
46         var constants = _links.Constants;
47         WriteHandlerState<TLink> handlerResult = new(constants.Continue, constants.Break,
48             ↪ handler);
49         handlerResult.Apply(_links.EnforceResetValues(linkIndex, handlerResult.Handler));
50         handlerResult.Apply(_links.Delete(restriction, handlerResult.Handler));
51         return handlerResult.Result;
52     }
53 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5  using TLink = System.UInt32;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Decorators
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the int 32 links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksDisposableDecoratorBase{TLink}"/>
18     public class UInt32Links : LinksDisposableDecoratorBase<TLink>
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt32Links"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public UInt32Links(ILinks<TLink> links) : base(links) { }
32
33         /// <summary>
34         /// <para>
35         /// Creates the substitution.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="substitution">
40         /// <para>The substitution.</para>
41         /// <para></para>
42         /// </param>
43         /// <returns>
44         /// <para>The link</para>
45         /// <para></para>
46         /// </returns>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public override TLink Create(ICollection<TLink>? substitution, WriteHandler<TLink>? handler)
49             ↪ => _links.CreatePoint(handler);
50
51         /// <summary>
52         /// <para>
53         /// Updates the substitution.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         /// <param name="restriction">
58         /// <para>The substitution.</para>
59         /// <para></para>
60         /// </param>
61         /// <param name="substitution">
62         /// <para>The substitution.</para>
63         /// <para></para>
64         /// </param>
65         /// <returns>

```



```

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

```

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

```

1  using System.Collections.Generic;
2  using System.Net.Security;
3  using System.Runtime.CompilerServices;
4  using Platform.Delegates;
5  using TLink = System.UInt64;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Decorators
10 {
11     /// <summary>
12     /// <para>Represents a combined decorator that implements the basic logic for interacting
13     ↪ with the links storage for links with addresses represented as <see cref="System.UInt64"
14     ↪ />.</para>

```

```

13  /// <para>Представляет комбинированный декоратор, реализующий основную логику по
    ↳ взаимодействию с хранилищем связей, для связей с адресами представленными в виде <see
    ↳ cref="System.UInt64"/>.</para>
14  /// </summary>
15  /// <remarks>
16  /// Возможные оптимизации:
17  /// Объединение в одном поле Source и Target с уменьшением до 32 бит.
18  /// + меньше объём БД
19  /// - меньше производительность
20  /// - больше ограничение на количество связей в БД)
21  /// Ленивое хранение размеров поддеревьев (расчитываемое по мере использования БД)
22  /// + меньше объём БД
23  /// - больше сложность
24  ///
25  /// Текущее теоретическое ограничение на индекс связи, из-за использования 5 бит в размере
    ↳ поддеревьев для AVL баланса и флагов нитей: 2 в степени(64 минус 5 равно 59 ) равно 576
    ↳ 460 752 303 423 488
26  /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами
    ↳ (битовыми строками) - вариант матрицы (выстраиваемой лениво).
27  ///
28  /// Решить отключать ли проверки при компиляции под Release. Т.е. исключения будут
    ↳ выбрасываться только при #if DEBUG
29  /// </remarks>
30  public class UInt64Links : LinksDisposableDecoratorBase<TLink>
31  {
32      /// <summary>
33      /// <para>
34      /// Initializes a new <see cref="UInt64Links"/> instance.
35      /// </para>
36      /// <para></para>
37      /// </summary>
38      /// <param name="links">
39      /// <para>A links.</para>
40      /// <para></para>
41      /// </param>
42      [MethodImpl(MethodImplOptions.AggressiveInlining)]
43      public UInt64Links(ILinks<TLink> links) : base(links) { }
44
45      /// <summary>
46      /// <para>
47      /// Creates the substitution.
48      /// </para>
49      /// <para></para>
50      /// </summary>
51      /// <param name="substitution">
52      /// <para>The substitution.</para>
53      /// <para></para>
54      /// </param>
55      /// <returns>
56      /// <para>The TLink</para>
57      /// <para></para>
58      /// </returns>
59      [MethodImpl(MethodImplOptions.AggressiveInlining)]
60      public override TLink Create(IList<TLink>? substitution, WriteHandler<TLink>? handler)
        ↳ => _links.CreatePoint(handler);
61
62      /// <summary>
63      /// <para>
64      /// Updates the substitution.
65      /// </para>
66      /// <para></para>
67      /// </summary>
68      /// <param name="restriction">
69      /// <para>The substitution.</para>
70      /// <para></para>
71      /// </param>
72      /// <param name="substitution">
73      /// <para>The substitution.</para>
74      /// <para></para>
75      /// </param>
76      /// <returns>
77      /// <para>The TLink</para>
78      /// <para></para>
79      /// </returns>
80      [MethodImpl(MethodImplOptions.AggressiveInlining)]
81      public override TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
        ↳ WriteHandler<TLink>? handler)

```

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using Platform.Collections;
5  using Platform.Collections.Lists;
6  using Platform.Data.Universal;
7  using Platform.Delegates;
8
9  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Decorators
12 {
13     /// <remarks>
14     /// What does empty pattern (for condition or substitution) mean? Nothing or Everything?
15     /// Now we go with nothing. And nothing is something one, but empty, and cannot be changed
16     ↪ by itself. But can cause creation (update from nothing) or deletion (update to nothing).
17     ///
18     /// TODO: Decide to change to IDoubletLinks or not to change. (Better to create
19     ↪ DefaultUniLinksBase, that contains logic itself and can be implemented using both
20     ↪ IDoubletLinks and ILinks.)

```

```

18  /// </remarks>
19  internal class UniLinks<TLink> : LinksDecoratorBase<TLink>, IUniLinks<TLink>
20  {
21      private static readonly EqualityComparer<TLink> _equalityComparer =
22          ↳ EqualityComparer<TLink>.Default;
23
24      /// <summary>
25      /// <para>
26      /// Initializes a new <see cref="UniLinks"/> instance.
27      /// </para>
28      /// </summary>
29      /// <param name="links">
30      /// <para>A links.</para>
31      /// </param>
32      public UniLinks(ILinks<TLink> links) : base(links) { }
33      private struct Transition
34      {
35          /// <summary>
36          /// <para>
37          /// The before.
38          /// </para>
39          /// </summary>
40          public IList<TLink>? Before;
41          /// <summary>
42          /// <para>
43          /// The after.
44          /// </para>
45          /// </summary>
46          public IList<TLink>? After;
47
48          /// <summary>
49          /// <para>
50          /// Initializes a new <see cref="Transition"/> instance.
51          /// </para>
52          /// </summary>
53          /// <param name="before">
54          /// <para>A before.</para>
55          /// </param>
56          /// <param name="after">
57          /// <para>A after.</para>
58          /// </param>
59          public Transition(IList<TLink>? before, IList<TLink>? after)
60          {
61              Before = before;
62              After = after;
63          }
64      }
65
66      //public static readonly TLink NullConstant = Use<LinksConstants<TLink>>.Single.Null;
67      //public static readonly IReadOnlyList<TLink> NullLink = new
68      ↳ ReadOnlyCollection<TLink>(new List<TLink> { NullConstant, NullConstant, NullConstant
69      ↳ });
70
71      // TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
72      ↳ (Links-Expression)
73      /// <summary>
74      /// <para>
75      /// Triggers the restriction.
76      /// </para>
77      /// </summary>
78      /// <param name="restriction">
79      /// <para>The restriction.</para>
80      /// </param>
81      /// <param name="matchedHandler">
82      /// <para>The matched handler.</para>
83      /// </param>
84      /// <param name="substitution">
85      /// <para>The substitution.</para>

```

```

92  /// <para></para>
93  /// </param>
94  /// <param name="substitutedHandler">
95  /// <para>The substituted handler.</para>
96  /// <para></para>
97  /// </param>
98  /// <returns>
99  /// <para>The link</para>
100 /// <para></para>
101 /// </returns>
102 public TLink Trigger(IList<TLink>? restriction, WriteHandler<TLink>? matchedHandler,
    ↳ IList<TLink>? substitution, WriteHandler<TLink>? substitutedHandler)
103 {
104     ///List<Transition> transitions = null;
105     ///if (!restriction.IsNullOrEmpty())
106     ///{
107     ///    // Есть причина делать проход (чтение)
108     ///    if (matchedHandler != null)
109     ///    {
110     ///        if (!substitution.IsNullOrEmpty())
111     ///        {
112     ///            // restriction => { 0, 0, 0 } | { 0 } // Create
113     ///            // substitution => { itself, 0, 0 } | { itself, itself, itself } //
    ↳ Create / Update
114     ///            // substitution => { 0, 0, 0 } | { 0 } // Delete
115     ///            transitions = new List<Transition>();
116     ///            if (Equals(substitution[Constants.IndexPart], Constants.Null))
117     ///            {
118     ///                // If index is Null, that means we always ignore every other
    ↳ value (they are also Null by definition)
119     ///                var matchDecision = matchedHandler(, NullLink);
120     ///                if (Equals(matchDecision, Constants.Break))
121     ///                    return false;
122     ///                if (!Equals(matchDecision, Constants.Skip))
123     ///                    transitions.Add(new Transition(matchedLink, newValue));
124     ///            }
125     ///            else
126     ///            {
127     ///                Func<T, bool> handler;
128     ///                handler = link =>
129     ///                {
130     ///                    var matchedLink = Memory.GetLinkValue(link);
131     ///                    var newValue = Memory.GetLinkValue(link);
132     ///                    newValue[Constants.IndexPart] = Constants.Itself;
133     ///                    newValue[Constants.SourcePart] =
    ↳ Equals(substitution[Constants.SourcePart], Constants.Itself) ?
    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.SourcePart];
134     ///                    newValue[Constants.TargetPart] =
    ↳ Equals(substitution[Constants.TargetPart], Constants.Itself) ?
    ↳ matchedLink[Constants.IndexPart] : substitution[Constants.TargetPart];
135     ///                    var matchDecision = matchedHandler(matchedLink, newValue);
136     ///                    if (Equals(matchDecision, Constants.Break))
137     ///                        return false;
138     ///                    if (!Equals(matchDecision, Constants.Skip))
139     ///                        transitions.Add(new Transition(matchedLink, newValue));
140     ///                    return true;
141     ///                };
142     ///                if (!Memory.Each(handler, restriction))
143     ///                    return Constants.Break;
144     ///            }
145     ///        }
146     ///    }
147     ///    else
148     ///    {
149     ///        Func<T, bool> handler = link =>
150     ///        {
151     ///            var matchedLink = Memory.GetLinkValue(link);
152     ///            var matchDecision = matchedHandler(matchedLink, matchedLink);
153     ///            return !Equals(matchDecision, Constants.Break);
154     ///        };
155     ///        if (!Memory.Each(handler, restriction))
156     ///            return Constants.Break;
157     ///    }
158     ///    else
159     ///    {
160     ///        if (substitution != null)
161     ///        {

```

```

162         transitions = new List<ILink<T>>>();
163         Func<T, bool> handler = link =>
164         {
165             var matchedLink = Memory.GetLinkValue(link);
166             transitions.Add(matchedLink);
167             return true;
168         };
169         if (!Memory.Each(handler, restriction))
170             return Constants.Break;
171     }
172     else
173     {
174         return Constants.Continue;
175     }
176 }
177 }
178 }
179 if (substitution != null)
180 {
181     // Есть причина делать замену (запись)
182     if (substitutedHandler != null)
183     {
184     }
185     else
186     {
187     }
188 }
189 }
190 return Constants.Continue;
191
192 //if (restriction.IsNullOrEmpty()) // Create
193 //{
194 //    substitution[Constants.IndexPart] = Memory.AllocateLink();
195 //    Memory.SetLinkValue(substitution);
196 //}
197 //else if (substitution.IsNullOrEmpty()) // Delete
198 //{
199 //    Memory.FreeLink(restriction[Constants.IndexPart]);
200 //}
201 //else if (restriction.EqualTo(substitution)) // Read or ("repeat" the state) // Each
202 //{
203 //    // No need to collect links to list
204 //    // Skip == Continue
205 //    // No need to check substitutedHandler
206 //    if (!Memory.Each(link => !Equals(matchedHandler(Memory.GetLinkValue(link)),
207 //        ↪ Constants.Break), restriction))
208 //        return Constants.Break;
209 //}
210 //else // Update
211 //{
212 //    //List<ILink<T>> matchedLinks = null;
213 //    if (matchedHandler != null)
214 //    {
215 //        matchedLinks = new List<ILink<T>>>();
216 //        Func<T, bool> handler = link =>
217 //        {
218 //            var matchedLink = Memory.GetLinkValue(link);
219 //            var matchDecision = matchedHandler(matchedLink);
220 //            if (Equals(matchDecision, Constants.Break))
221 //                return false;
222 //            if (!Equals(matchDecision, Constants.Skip))
223 //                matchedLinks.Add(matchedLink);
224 //            return true;
225 //        };
226 //        if (!Memory.Each(handler, restriction))
227 //            return Constants.Break;
228 //    }
229 //    if (!matchedLinks.IsNullOrEmpty())
230 //    {
231 //        var totalMatchedLinks = matchedLinks.Count;
232 //        for (var i = 0; i < totalMatchedLinks; i++)
233 //        {
234 //            var matchedLink = matchedLinks[i];
235 //            if (substitutedHandler != null)
236 //            {
237 //                var newValue = new List<T>(); // TODO: Prepare value to update here
238 //                // TODO: Decide is it actually needed to use Before and After
239 //                ↪ substitution handling.
240 //                var substitutedDecision = substitutedHandler(matchedLink,
241 //                ↪ newValue);

```

```

237         //         if (Equals(substitutedDecision, Constants.Break))
238         //             return Constants.Break;
239         //         if (Equals(substitutedDecision, Constants.Continue))
240         //         {
241         //             // Actual update here
242         //             Memory.SetLinkValue(newValue);
243         //         }
244         //         if (Equals(substitutedDecision, Constants.Skip))
245         //         {
246         //             // Cancel the update. TODO: decide use separate Cancel
247         //             ↪ constant or Skip is enough?
248         //         }
249         //     }
250     }
251 }
252 return _constants.Continue;
253 }
254
255 /// <summary>
256 /// <para>
257 /// Triggers the pattern or condition.
258 /// </para>
259 /// <para></para>
260 /// </summary>
261 /// <param name="patternOrCondition">
262 /// <para>The pattern or condition.</para>
263 /// <para></para>
264 /// </param>
265 /// <param name="matchHandler">
266 /// <para>The match handler.</para>
267 /// <para></para>
268 /// </param>
269 /// <param name="substitution">
270 /// <para>The substitution.</para>
271 /// <para></para>
272 /// </param>
273 /// <param name="substitutionHandler">
274 /// <para>The substitution handler.</para>
275 /// <para></para>
276 /// </param>
277 /// <exception cref="NotImplementedException">
278 /// <para></para>
279 /// <para></para>
280 /// </exception>
281 /// <exception cref="NotSupportedException">
282 /// <para></para>
283 /// <para></para>
284 /// </exception>
285 /// <exception cref="NotSupportedException">
286 /// <para></para>
287 /// <para></para>
288 /// </exception>
289 /// <exception cref="NotSupportedException">
290 /// <para></para>
291 /// <para></para>
292 /// </exception>
293 /// <exception cref="NotSupportedException">
294 /// <para></para>
295 /// <para></para>
296 /// </exception>
297 /// <returns>
298 /// <para>The link</para>
299 /// <para></para>
300 /// </returns>
301 public TLink Trigger(IList<TLink>? patternOrCondition, ReadHandler<TLink>? matchHandler,
302     ↪ IList<TLink>? substitution, WriteHandler<TLink>? substitutionHandler)
303 {
304     var constants = _constants;
305     if (patternOrCondition.IsNullOrEmpty() && substitution.IsNullOrEmpty())
306     {
307         return constants.Continue;
308     }
309     else if (patternOrCondition.EqualTo(substitution)) // Should be Each here TODO:
310     ↪ Check if it is a correct condition
311     {
312         // Or it only applies to trigger without matchHandler.
313         throw new NotImplementedException();

```

```

312 }
313 else if (!substitution.IsNullOrEmpty()) // Creation
314 {
315     var before = Array.Empty<TLink>();
316     // Что должно означать False здесь? Остановиться (перестать идти) или пропустить
317     ↪ (пройти мимо) или пустить (взять)?
318     if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
319     ↪ constants.Break))
320     {
321         return constants.Break;
322     }
323     var after = (IList<TLink>?)substitution.ToArray();
324     if (_equalityComparer.Equals(after[0], default))
325     {
326         var newLink = _links.Create();
327         after[0] = newLink;
328     }
329     if (substitution.Count == 1)
330     {
331         after = _links.GetLink(substitution[0]);
332     }
333     else if (substitution.Count == 3)
334     {
335         //Links.Create(after);
336     }
337     else
338     {
339         throw new NotSupportedException();
340     }
341     return matchHandler != null ? substitutionHandler(before, after) :
342     ↪ constants.Continue;
343 }
344 else if (!patternOrCondition.IsNullOrEmpty()) // Deletion
345 {
346     if (patternOrCondition.Count == 1)
347     {
348         var linkToDelete = patternOrCondition[0];
349         var before = _links.GetLink(linkToDelete);
350         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
351         ↪ constants.Break))
352         {
353             return constants.Break;
354         }
355         var after = Array.Empty<TLink>();
356         _links.Update(linkToDelete, constants.Null, constants.Null);
357         _links.Delete(linkToDelete);
358         return matchHandler != null ? substitutionHandler(before, after) :
359         ↪ constants.Continue;
360     }
361     else
362     {
363         throw new NotSupportedException();
364     }
365 }
366 else // Replace / Update
367 {
368     if (patternOrCondition.Count == 1) //-V3125
369     {
370         var linkToUpdate = patternOrCondition[0];
371         var before = _links.GetLink(linkToUpdate);
372         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
373         ↪ constants.Break))
374         {
375             return constants.Break;
376         }
377         var after = (IList<TLink>?)substitution.ToArray(); //-V3125
378         if (_equalityComparer.Equals(after[0], default))
379         {
380             after[0] = linkToUpdate;
381         }
382         if (substitution.Count == 1)
383         {
384             if (!_equalityComparer.Equals(substitution[0], linkToUpdate))
385             {
386                 after = _links.GetLink(substitution[0]);
387                 _links.Update(linkToUpdate, constants.Null, constants.Null);
388                 _links.Delete(linkToUpdate);
389             }
390         }
391     }
392 }

```



```

384     }
385     else if (substitution.Count == 3)
386     {
387         //Links.Update(after);
388     }
389     else
390     {
391         throw new NotSupportedException();
392     }
393     return matchHandler != null ? substitutionHandler(before, after) :
        ↪ constants.Continue;
394 }
395 else
396 {
397     throw new NotSupportedException();
398 }
399 }
400 }
401
402 /// <remarks>
403 /// IList[IList[IList[T]]]
404 /// | | | |
405 /// | | | |-----| |
406 /// | | | | link | |
407 /// | | | |-----| |
408 /// | | | | change | |
409 /// |-----|
410 /// | changes
411 /// </remarks>
412 public IList<IList<IList<TLink>?>> Trigger(IList<TLink>? condition, IList<TLink>?
    ↪ substitution)
413 {
414     var changes = new List<IList<IList<TLink>?>>();
415     var @continue = _constants.Continue;
416     Trigger(condition, AlwaysContinue, substitution, (before, after) =>
417     {
418         var change = new[] { before, after };
419         changes.Add(change);
420         return @continue;
421     });
422     return changes;
423 }
424 private TLink AlwaysContinue(IList<TLink>? linkToMatch) => _constants.Continue;
425 }
426 }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets
8 {
9
10     /// <summary>
11     /// <para>.</para>
12     /// <para>.</para>
13     /// </summary>
14     /// <typeparam>
15     /// <para>.</para>
16     /// <para>.</para>
17     /// </typeparam>
18     public struct Doublet<T> : IEquatable<Doublet<T>>
19     {
20         private static readonly EqualityComparer<T> _equalityComparer =
            ↪ EqualityComparer<T>.Default;
21
22         /// <summary>
23         /// <para>.</para>
24         /// <para>.</para>
25         /// </summary>
26         /// <typeparam name="T">
27         /// <para>.</para>
28         /// <para>.</para>
29         /// </typeparam>
30         public readonly T Source;
31

```

```

32    /// <summury>
33    /// <para>.</para>
34    /// <para>.</para>
35    /// </summury>
36    /// <typeparam name="T">
37    /// <para>.</para>
38    /// <para>.</para>
39    /// </typeparam>
40    public readonly T Target;
41
42    /// <summury>
43    /// <para>.</para>
44    /// <para>.</para>
45    /// </summury>
46    /// <typeparam name="T">
47    /// <para>.</para>
48    /// <para>.</para>
49    /// </typeparam>
50    /// <param name="source">
51    /// <para>.</para>
52    /// <para>.</para>
53    /// </param>
54    /// <param name="target">
55    /// <para>.</para>
56    /// <para>.</para>
57    /// </param>
58    [MethodImpl(MethodImplOptions.AggressiveInlining)]
59    public Doublet(T source, T target)
60    {
61        Source = source;
62        Target = target;
63    }
64
65    /// <summury>
66    /// <para>.</para>
67    /// <para>.</para>
68    /// </summury>
69    /// <returns>
70    /// <para>.</para>
71    /// <para>.</para>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    public override string ToString() => $"{Source}->{Target}";
75
76    /// <summury>
77    /// <para>.</para>
78    /// <para>.</para>
79    /// </summury>
80    /// <typeparam>
81    /// <para>.</para>
82    /// <para>.</para>
83    /// </typeparam>
84    /// <param name="other">
85    /// <para>.</para>
86    /// <para>.</para>
87    /// </param>
88    /// <returns>
89    /// <para>.</para>
90    /// <para>.</para>
91    /// </returns>
92    [MethodImpl(MethodImplOptions.AggressiveInlining)]
93    public bool Equals(Doublet<T> other) => _equalityComparer.Equals(Source, other.Source)
94    ↪ && _equalityComparer.Equals(Target, other.Target);
95
96    /// <summury>
97    /// <para>.</para>
98    /// <para>.</para>
99    /// </summury>
100    /// <typeparam>
101    /// <para>.</para>
102    /// <para>.</para>
103    /// </typeparam>
104    /// <param name="obj">
105    /// <para>.</para>
106    /// <para>.</para>
107    /// </param>
108    /// <returns>
109    /// <para>.</para>

```

```

109     /// <para>.</para>
110     /// </returns>
111     [MethodImpl(MethodImplOptions.AggressiveInlining)]
112     public override bool Equals(object obj) => obj is Doublet<T> doublet ?
        ↳ base.Equals(doublet) : false;
113
114     /// <summary>
115     /// <para>.</para>
116     /// <para>.</para>
117     /// </summary>
118     /// <returns>
119     /// <para>.</para>
120     /// <para>.</para>
121     /// </returns>
122     [MethodImpl(MethodImplOptions.AggressiveInlining)]
123     public override int GetHashCode() => (Source, Target).GetHashCode();
124
125     /// <summary>
126     /// <para>.</para>
127     /// <para>.</para>
128     /// </summary>
129     /// <param name="left">
130     /// <para>.</para>
131     /// <para>.</para>
132     /// </param>
133     /// <param name="right">
134     /// <para>.</para>
135     /// <para>.</para>
136     /// </param>
137     /// <returns>
138     /// <para>.</para>
139     /// <para>.</para>
140     /// </returns>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static bool operator ==(Doublet<T> left, Doublet<T> right) => left.Equals(right);
143
144     /// <summary>
145     /// <para>.</para>
146     /// <para>.</para>
147     /// </summary>
148     /// <param name="left">
149     /// <para>.</para>
150     /// <para>.</para>
151     /// </param>
152     /// <param name="right">
153     /// <para>.</para>
154     /// <para>.</para>
155     /// </param>
156     /// <returns>
157     /// <para>.</para>
158     /// <para>.</para>
159     /// </returns>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public static bool operator !=(Doublet<T> left, Doublet<T> right) => !(left == right);
162 }
163 }

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets
7  {
8      /// <remarks>
9      /// TODO: Может стоит попробовать ref во всех методах (IRefEqualityComparer)
10     /// 2x faster with comparer
11     /// </remarks>
12     public class DoubletComparer<T> : IEqualityComparer<Doublet<T>>
13     {
14         /// <summary>
15         /// <para>
16         /// The .
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         public static readonly DoubletComparer<T> Default = new DoubletComparer<T>();

```

```

21
22     /// <summary>
23     /// <para>
24     /// Determines whether this instance equals.
25     /// </para>
26     /// <para></para>
27     /// </summary>
28     /// <param name="x">
29     /// <para>The .</para>
30     /// <para></para>
31     /// </param>
32     /// <param name="y">
33     /// <para>The .</para>
34     /// <para></para>
35     /// </param>
36     /// <returns>
37     /// <para>The bool</para>
38     /// <para></para>
39     /// </returns>
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     public bool Equals(Doublet<T> x, Doublet<T> y) => x.Equals(y);
42
43     /// <summary>
44     /// <para>
45     /// Gets the hash code using the specified obj.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="obj">
50     /// <para>The obj.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>The int</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public int GetHashCode(Doublet<T> obj) => obj.GetHashCode();
59 }
60 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Runtime.InteropServices;
5  using Platform.Converters;
6  using Platform.Delegates;
7  using Platform.Disposables;
8
9  namespace Platform.Data.Doublets.FFI
10 {
11     using TLink = System.UInt32;
12
13     public class UInt32UnitedMemoryLinks : DisposableBase, ILinks<TLink>
14     {
15         public LinksConstants<TLink> Constants { get; }
16
17         private readonly unsafe void* _ptr;
18
19         public UInt32UnitedMemoryLinks(string path)
20         {
21             unsafe
22             {
23                 _ptr = Methods.UInt32UnitedMemoryLinks_New(path);
24
25                 // TODO: Update api
26                 Constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
27             }
28         }
29
30         public TLink Count(ICollection<TLink>? restriction)
31         {
32             unsafe
33             {
34                 var array = stackalloc uint[restriction.Count];
35                 for (var i = 0; i < restriction.Count; i++)
36                 {
37                     array[i] = restriction[i];
38                 }
39             }
40         }
41     }
42 }

```

```

38     }
39     return Methods.UInt32UnitedMemoryLinks_Count(_ptr, array,
    ↪ (nuint)(restriction?.Count ?? 0));
40 }
41 }
42
43 public TLink Each(IList<TLink>? restriction, ReadHandler<TLink>? handler)
44 {
45     unsafe
46     {
47         Methods.EachCallback UInt32 callback = (link) => handler?.Invoke(new
    ↪ Link<TLink>(link.Index, link.Source, link.Target)) ?? Constants.Continue;
48         var array = stackalloc uint[restriction.Count];
49         for (var i = 0; i < restriction.Count; i++)
50         {
51             array[i] = restriction[i];
52         }
53         return Methods.UInt32UnitedMemoryLinks_Each(_ptr, array,
    ↪ (nuint)(restriction?.Count ?? 0), callback);
54     }
55 }
56
57 public TLink Create(IList<TLink>? substitution, WriteHandler<TLink>? handler)
58 {
59     unsafe
60     {
61         Methods.CreateCallback UInt32 callback = (before, after) => handler?.Invoke(new
    ↪ Link<TLink>(before.Index, before.Source, before.Target), new
    ↪ Link<TLink>(after.Index, after.Source, after.Target)) ?? Constants.Continue;
62         fixed (uint* substitutionPtr = (uint[])substitution)
63         {
64             return Methods.UInt32UnitedMemoryLinks_Create(_ptr, substitutionPtr,
    ↪ (nuint)(substitution?.Count ?? 0), callback);
65         }
66     }
67 }
68
69 public TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
    ↪ WriteHandler<TLink>? handler)
70 {
71     unsafe
72     {
73         var restrictionArray = stackalloc uint[restriction.Count];
74         for (var i = 0; i < restriction.Count; i++)
75         {
76             restrictionArray[i] = restriction[i];
77         }
78         var substitutionArray = stackalloc uint[substitution.Count];
79         for (var i = 0; i < restriction.Count; i++)
80         {
81             substitutionArray[i] = restriction[i];
82         }
83         Methods.UpdateCallback UInt32 callback = (before, after) => handler?.Invoke(new
    ↪ Link<TLink>(before.Index, before.Source, before.Target), new
    ↪ Link<TLink>(after.Index, after.Source, after.Target)) ?? Constants.Continue;
84         return Methods.UInt32UnitedMemoryLinks_Update(_ptr, restrictionArray,
    ↪ (nuint)(restriction?.Count ?? 0), substitutionArray,
    ↪ (nuint)(substitution?.Count ?? 0), callback);
85     }
86 }
87
88 public TLink Delete(IList<TLink>? restriction, WriteHandler<TLink>? handler)
89 {
90     unsafe
91     {
92         var restrictionArray = stackalloc uint[restriction.Count];
93         for (var i = 0; i < restriction.Count; i++)
94         {
95             restrictionArray[i] = restriction[i];
96         }
97         Methods.DeleteCallback UInt32 callback = (before, after) => handler?.Invoke(new
    ↪ Link<TLink>(before.Index, before.Source, before.Target), new
    ↪ Link<TLink>(after.Index, after.Source, after.Target)) ?? Constants.Continue;
98         return Methods.UInt32UnitedMemoryLinks_Delete(_ptr, restrictionArray,
    ↪ (nuint)(restriction?.Count ?? 0), callback);
99     }
100 }

```

```

101     protected override void Dispose(bool manual, bool wasDisposed)
102     {
103         unsafe
104         {
105             if (wasDisposed || _ptr == null)
106             {
107                 return;
108             }
109             Methods.UInt32UnitedMemoryLinks_Drop(_ptr);
110         }
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     }
67 }

```

```

64 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
65 public delegate UInt64 CreateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
    ↪ after);
66
67 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
68 public delegate Byte UpdateCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
69
70 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
71 public delegate UInt16 UpdateCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
    ↪ after);
72
73 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
74 public delegate UInt32 UpdateCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
    ↪ after);
75
76 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
77 public delegate UInt64 UpdateCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
    ↪ after);
78
79 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
80 public delegate Byte DeleteCallback_UInt8(FfiLink_UInt8 before, FfiLink_UInt8 after);
81
82 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
83 public delegate UInt16 DeleteCallback_UInt16(FfiLink_UInt16 before, FfiLink_UInt16
    ↪ after);
84
85 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
86 public delegate UInt32 DeleteCallback_UInt32(FfiLink_UInt32 before, FfiLink_UInt32
    ↪ after);
87
88 [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
89 public delegate UInt64 DeleteCallback_UInt64(FfiLink_UInt64 before, FfiLink_UInt64
    ↪ after);
90
91 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
92 public static extern void* ByteUnitedMemoryLinks_New(string path);
93
94 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
95 public static extern void* UInt16UnitedMemoryLinks_New(string path);
96
97 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
98 public static extern void* UInt32UnitedMemoryLinks_New(string path);
99
100 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
101 public static extern void* UInt64UnitedMemoryLinks_New(string path);
102
103 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
104 public static extern void ByteUnitedMemoryLinks_Drop(void* self);
105
106 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
107 public static extern void UInt16UnitedMemoryLinks_Drop(void* self);
108
109 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
110 public static extern void UInt32UnitedMemoryLinks_Drop(void* self);
111
112 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
113 public static extern void UInt64UnitedMemoryLinks_Drop(void* self);
114
115 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
116 public static extern byte ByteUnitedMemoryLinks_Create(void* self, byte* substitution,
    ↪ nuint substitutionLength, CreateCallback_UInt8 callback);
117
118 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
119 public static extern ushort UInt16UnitedMemoryLinks_Create(void* self, ushort*
    ↪ substitution, nuint substitutionLength, CreateCallback_UInt16 callback);
120
121 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
122 public static extern uint UInt32UnitedMemoryLinks_Create(void* self, uint* substitution,
    ↪ nuint substitutionLength, CreateCallback_UInt32 callback);
123
124 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
125 public static extern ulong UInt64UnitedMemoryLinks_Create(void* self, ulong*
    ↪ substitution, nuint substitutionLength, CreateCallback_UInt64 callback);
126
127 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
128 public static extern byte ByteUnitedMemoryLinks_Count(void* self, byte* restriction,
    ↪ nuint len);
129

```

```

130 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
131 public static extern ushort UInt16UnitedMemoryLinks_Count(void* self, ushort*
    ↳ restriction, nuint len);
132
133 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
134 public static extern uint UInt32UnitedMemoryLinks_Count(void* self, uint* restriction,
    ↳ nuint len);
135
136 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
137 public static extern ulong UInt64UnitedMemoryLinks_Count(void* self, ulong* restriction,
    ↳ nuint len);
138
139 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
140 public static extern byte ByteUnitedMemoryLinks_Each(void* self, byte* restriction,
    ↳ nuint len, EachCallback UInt8 callback);
141
142 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
143 public static extern ushort UInt16UnitedMemoryLinks_Each(void* self, ushort*
    ↳ restriction, nuint len, EachCallback UInt16 callback);
144
145 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
146 public static extern uint UInt32UnitedMemoryLinks_Each(void* self, uint* restriction,
    ↳ nuint len, EachCallback UInt32 callback);
147
148 [DllImport(DllName, CallingConvention = CallingConvention.Cdecl)]
149 public static extern ulong UInt64UnitedMemoryLinks_Each(void* self, ulong* restriction,
    ↳ nuint len, EachCallback UInt64 callback);
150
151 [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<TLink> : DisposableBase, ILinks<TLink> where TLink : struct
177 {
178     private static readonly UncheckedConverter<byte, TLink> from_u8 =
        ↳ UncheckedConverter<byte, TLink>.Default;
179     private static readonly UncheckedConverter<ushort, TLink> from_u16 =
        ↳ UncheckedConverter<ushort, TLink>.Default;
180     private static readonly UncheckedConverter<uint, TLink> from_u32 =
        ↳ UncheckedConverter<uint, TLink>.Default;
181     private static readonly UncheckedConverter<ulong, TLink> from_u64 =
        ↳ UncheckedConverter<ulong, TLink>.Default;
182     private static readonly UncheckedConverter<TLink, ulong> from_t =
        ↳ UncheckedConverter<TLink, ulong>.Default;
183

```



```

184 public LinksConstants<TLink> Constants { get; }
185
186 private readonly unsafe void* _ptr;
187
188 public UnitedMemoryLinks(string path)
189 {
190     TLink 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<TLink>(enableExternalReferencesSupport: true);
204     }
205 }
206
207 public TLink Count(IList<TLink>? restriction)
208 {
209     var restrictionLength = restriction?.Count ?? 0;
210     unsafe
211     {
212         TLink 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             {

```

```

259         throw new NotImplementedException();
260     }
261 }
262 }
263 }
264
265 public TLink Each(IList<TLink>? restriction, ReadHandler<TLink>? handler)
266 {
267     var restrictionLength = restriction?.Count ?? 0;
268     unsafe
269     {
270         TLink t = default;
271         switch (t)
272         {
273             case byte:
274             {
275                 byte Callback(FfiLink_UInt8 link) =>
276                 ↪ (byte)from_t.Convert(handler?.Invoke(new
277                 ↪ Link<TLink>(from_u8.Convert(link.Index),
278                 ↪ from_u8.Convert(link.Source), from_u8.Convert(link.Target))) ??
279                 ↪ Constants.Continue);
280                 var restrictionArray = stackalloc byte[restrictionLength];
281                 var byteRestrictionArray = (IList<byte>)restriction;
282                 for (var i = 0; i < restrictionLength; i++)
283                 {
284                     restrictionArray[i] = byteRestrictionArray[i];
285                 }
286                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Each(_ptr,
287                 ↪ restrictionArray, (nuint)restrictionLength, Callback));
288             }
289             case ushort:
290             {
291                 ushort Callback(FfiLink_UInt16 link) =>
292                 ↪ (ushort)from_t.Convert(handler?.Invoke(new
293                 ↪ Link<TLink>(from_u16.Convert(link.Index),
294                 ↪ from_u16.Convert(link.Source), from_u16.Convert(link.Target))) ??
295                 ↪ Constants.Continue);
296                 var restrictionArray = stackalloc ushort[restrictionLength];
297                 var ushortRestrictionArray = (IList<ushort>)restriction;
298                 for (var i = 0; i < restrictionLength; i++)
299                 {
300                     restrictionArray[i] = ushortRestrictionArray[i];
301                 }
302                 return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Each(_ptr,
303                 ↪ restrictionArray, (nuint)restrictionLength, Callback));
304             }
305             case uint:
306             {
307                 uint Callback(FfiLink_UInt32 link) =>
308                 ↪ (uint)from_t.Convert(handler?.Invoke(new
309                 ↪ Link<TLink>(from_u32.Convert(link.Index),
310                 ↪ from_u32.Convert(link.Source), from_u32.Convert(link.Target))) ??
311                 ↪ Constants.Continue);
312                 var restrictionArray = stackalloc uint[restrictionLength];
313                 var uintRestrictionArray = (IList<uint>)restriction;
314                 for (var i = 0; i < restrictionLength; i++)
315                 {
316                     restrictionArray[i] = uintRestrictionArray[i];
317                 }
318                 return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Each(_ptr,
319                 ↪ restrictionArray, (nuint)restrictionLength, Callback));
320             }
321             case ulong:
322             {
323                 {
324                     ulong Callback(FfiLink_UInt64 link) =>
325                     ↪ from_t.Convert(handler?.Invoke(new
326                     ↪ Link<TLink>(from_u64.Convert(link.Index),
327                     ↪ from_u64.Convert(link.Source), from_u64.Convert(link.Target)))
328                     ↪ ?? Constants.Continue);
329                     var restrictionArray = stackalloc UInt64[restrictionLength];
330                     var ulongRestrictionArray = (IList<ulong>)restriction;
331                     for (var i = 0; i < restrictionLength; i++)
332                     {
333                         restrictionArray[i] = ulongRestrictionArray[i];
334                     }
335                     return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Each(_ptr,
336                     ↪ restrictionArray, (nuint)restrictionLength, Callback));
337                 }
338             }
339         }
340     }
341 }

```

```

317     }
318 }
319 default:
320 {
321     throw new NotImplementedException();
322 }
323 }
324 }
325 }
326
327 public TLink Create(ICollection<TLink>? substitution, WriteHandler<TLink>? handler)
328 {
329     var substitutionLength = substitution?.Count ?? 0;
330     unsafe
331     {
332         TLink 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<TLink>(from_u8.Convert(before.Index),
340                     ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
341                     ↪ Link<TLink>(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 = (ICollection<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<TLink>(from_u16.Convert(before.Index),
358                     ↪ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
359                     ↪ new Link<TLink>(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 = (ICollection<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<TLink>(from_u32.Convert(before.Index),
376                     ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
377                     ↪ new Link<TLink>(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 = (ICollection<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             {

```

```

371         ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
372             (ulong)from_t.Convert(handler?.Invoke(new
373                 ↳ Link<TLink>(from_u64.Convert(before.Index),
374                 ↳ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
375                 ↳ new Link<TLink>(from_u64.Convert(after.Index),
376                 ↳ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) ??
377                 ↳ Constants.Continue);
378         var substitutionArray = stackalloc ulong[substitutionLength];
379         var ulongSubstitutionArray = (IList<ulong>)substitution;
380         for (var i = 0; i < substitutionLength; i++)
381         {
382             substitutionArray[i] = ulongSubstitutionArray[i];
383         }
384         return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Create(_ptr,
385             ↳ substitutionArray, (nuint)(substitution?.Count ?? 0), Callback));
386     }
387     default:
388     {
389         throw new NotImplementedException();
390     }
391 };
392 }
393
394 public TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
395     ↳ WriteHandler<TLink>? handler)
396 {
397     var restrictionLength = restriction?.Count ?? 0;
398     var substitutionLength = substitution?.Count ?? 0;
399     unsafe
400     {
401         TLink t = default;
402         switch (t)
403         {
404             case byte:
405             {
406                 var restrictionArray = stackalloc byte[restrictionLength];
407                 var byteRestrictionArray = (IList<byte>)restriction;
408                 for (var i = 0; i < restrictionLength; i++)
409                 {
410                     restrictionArray[i] = byteRestrictionArray[i];
411                 }
412                 var substitutionArray = stackalloc byte[substitutionLength];
413                 var byteSubstitutionArray = (IList<byte>)substitution;
414                 for (var i = 0; i < substitutionLength; i++)
415                 {
416                     substitutionArray[i] = byteSubstitutionArray[i];
417                 }
418                 byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
419                     (byte)from_t.Convert(handler?.Invoke(new
420                         ↳ Link<TLink>(from_u8.Convert(before.Index),
421                         ↳ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
422                         ↳ Link<TLink>(from_u8.Convert(after.Index),
423                         ↳ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) ??
424                         ↳ Constants.Continue);
425                 return from_u8.Convert(Methods.ByteUnitedMemoryLinks_Update(_ptr,
426                     ↳ restrictionArray, (nuint)restrictionLength, substitutionArray,
427                     ↳ (nuint)(substitution?.Count ?? 0), Callback));
428             }
429             case ushort:
430             {
431                 var restrictionArray = stackalloc ushort[restrictionLength];
432                 var ushortRestrictionArray = (IList<ushort>)restriction;
433                 for (var i = 0; i < restrictionLength; i++)
434                 {
435                     restrictionArray[i] = ushortRestrictionArray[i];
436                 }
437                 var substitutionArray = stackalloc ushort[substitutionLength];
438                 var ushortSubstitutionArray = (IList<ushort>)substitution;
439                 for (var i = 0; i < substitutionLength; i++)
440                 {
441                     substitutionArray[i] = ushortSubstitutionArray[i];
442                 }
443             }
444         }
445     }
446 }

```

```

ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
    (ushort)from_t.Convert(handler?.Invoke(new
    Link<TLink>(from_u16.Convert(before.Index),
    from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
    new Link<TLink>(from_u16.Convert(after.Index),
    from_u16.Convert(after.Source), from_u16.Convert(after.Target))) ??
    Constants.Continue);
return from_u16.Convert(Methods.UInt16UnitedMemoryLinks_Update(_ptr,
    restrictionArray, (nuint)restrictionLength, substitutionArray,
    (nuint)(substitution?.Count ?? 0), Callback));
}
case uint:
{
    var restrictionArray = stackalloc uint[restrictionLength];
    var uintRestrictionArray = (IList<uint>)restriction;
    for (var i = 0; i < restrictionLength; i++)
    {
        restrictionArray[i] = uintRestrictionArray[i];
    }
    var substitutionArray = stackalloc uint[substitutionLength];
    var uintSubstitutionArray = (IList<uint>)substitution;
    for (var i = 0; i < substitutionLength; i++)
    {
        substitutionArray[i] = uintSubstitutionArray[i];
    }
    uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
        (uint)from_t.Convert(handler?.Invoke(new
        Link<TLink>(from_u32.Convert(before.Index),
        from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
        new Link<TLink>(from_u32.Convert(after.Index),
        from_u32.Convert(after.Source), from_u32.Convert(after.Target))) ??
        Constants.Continue);
    return from_u32.Convert(Methods.UInt32UnitedMemoryLinks_Update(_ptr,
        restrictionArray, (nuint)restrictionLength, substitutionArray,
        (nuint)(substitution?.Count ?? 0), Callback));
}
case ulong:
{
    var restrictionArray = stackalloc ulong[restrictionLength];
    var ulongRestrictionArray = (IList<ulong>)restriction;
    for (var i = 0; i < restrictionLength; i++)
    {
        restrictionArray[i] = ulongRestrictionArray[i];
    }
    var substitutionArray = stackalloc ulong[substitutionLength];
    var ulongSubstitutionArray = (IList<ulong>)substitution;
    for (var i = 0; i < substitutionLength; i++)
    {
        substitutionArray[i] = ulongSubstitutionArray[i];
    }
    ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
        (ulong)from_t.Convert(handler?.Invoke(new
        Link<TLink>(from_u64.Convert(before.Index),
        from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
        new Link<TLink>(from_u64.Convert(after.Index),
        from_u64.Convert(after.Source), from_u64.Convert(after.Target))) ??
        Constants.Continue);
    return from_u64.Convert(Methods.UInt64UnitedMemoryLinks_Update(_ptr,
        restrictionArray, (nuint)restrictionLength, substitutionArray,
        (nuint)(substitution?.Count ?? 0), Callback));
}
default:
{
    throw new NotImplementedException();
}
}
}
}

public TLink Delete(IList<TLink>? restriction, WriteHandler<TLink>? handler)
{
    var restrictionLength = restriction?.Count ?? 0;
    unsafe
    {
        TLink t = default;
        switch (t)
        {
            case byte:

```

```

483 {
484     var restrictionArray = stackalloc byte[restrictionLength];
485     var byteRestrictionArray = (IList<byte>)restriction;
486     for (var i = 0; i < restrictionLength; i++)
487     {
488         restrictionArray[i] = byteRestrictionArray[i];
489     }
490     byte Callback(FfiLink_UInt8 before, FfiLink_UInt8 after) =>
491     {
492         (byte)from_t.Convert(handler?.Invoke(new
493             ↪ Link<TLink>(from_u8.Convert(before.Index),
494             ↪ from_u8.Convert(before.Source), from_u8.Convert(before.Target)), new
495             ↪ Link<TLink>(from_u8.Convert(after.Index),
496             ↪ from_u8.Convert(after.Source), from_u8.Convert(after.Target))) ??
497             ↪ Constants.Continue);
498         return (TLink)(object)Methods.ByteUnitedMemoryLinks_Delete(_ptr,
499             ↪ restrictionArray, (nuint)restrictionLength, Callback);
500     }
501     case ushort:
502     {
503         var restrictionArray = stackalloc ushort[restrictionLength];
504         var ushortRestrictionArray = (IList<ushort>)restriction;
505         for (var i = 0; i < restrictionLength; i++)
506         {
507             restrictionArray[i] = ushortRestrictionArray[i];
508         }
509         ushort Callback(FfiLink_UInt16 before, FfiLink_UInt16 after) =>
510         {
511             (ushort)from_t.Convert(handler?.Invoke(new
512                 ↪ Link<TLink>(from_u16.Convert(before.Index),
513                 ↪ from_u16.Convert(before.Source), from_u16.Convert(before.Target)),
514                 ↪ new Link<TLink>(from_u16.Convert(after.Index),
515                 ↪ from_u16.Convert(after.Source), from_u16.Convert(after.Target))) ??
516                 ↪ Constants.Continue);
517             return (TLink)(object)Methods.UInt16UnitedMemoryLinks_Delete(_ptr,
518                 ↪ restrictionArray, (nuint)restrictionLength, Callback);
519         }
520     }
521     case uint:
522     {
523         var restrictionArray = stackalloc uint[restrictionLength];
524         var uintRestrictionArray = (IList<uint>)restriction;
525         for (var i = 0; i < restrictionLength; i++)
526         {
527             restrictionArray[i] = uintRestrictionArray[i];
528         }
529         uint Callback(FfiLink_UInt32 before, FfiLink_UInt32 after) =>
530         {
531             (uint)from_t.Convert(handler?.Invoke(new
532                 ↪ Link<TLink>(from_u32.Convert(before.Index),
533                 ↪ from_u32.Convert(before.Source), from_u32.Convert(before.Target)),
534                 ↪ new Link<TLink>(from_u32.Convert(after.Index),
535                 ↪ from_u32.Convert(after.Source), from_u32.Convert(after.Target))) ??
536                 ↪ Constants.Continue);
537             return (TLink)(object)Methods.UInt32UnitedMemoryLinks_Delete(_ptr,
538                 ↪ restrictionArray, (nuint)restrictionLength, Callback);
539         }
540     }
541     case ulong:
542     {
543         var restrictionArray = stackalloc ulong[restrictionLength];
544         var ulongRestrictionArray = (IList<ulong>)restriction;
545         for (var i = 0; i < restrictionLength; i++)
546         {
547             restrictionArray[i] = ulongRestrictionArray[i];
548         }
549         ulong Callback(FfiLink_UInt64 before, FfiLink_UInt64 after) =>
550         {
551             (ulong)from_t.Convert(handler?.Invoke(new
552                 ↪ Link<TLink>(from_u64.Convert(before.Index),
553                 ↪ from_u64.Convert(before.Source), from_u64.Convert(before.Target)),
554                 ↪ new Link<TLink>(from_u64.Convert(after.Index),
555                 ↪ from_u64.Convert(after.Source), from_u64.Convert(after.Target))) ??
556                 ↪ Constants.Continue);
557             return (TLink)(object)Methods.UInt64UnitedMemoryLinks_Delete(_ptr,
558                 ↪ restrictionArray, (nuint)restrictionLength, Callback);
559         }
560     }
561     default:
562     {
563         throw new NotImplementedException();
564     }
565 }

```

```

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             TLink 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{TLink, LinksConstants{TLink}}"/>
14     public interface ILinks<TLink> : ILinks<TLink, LinksConstants<TLink>>
15     {
16     }
17 }

```

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

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Runtime.CompilerServices;
6  using Platform.Ranges;
7  using Platform.Collections.Lists;
8  using Platform.Random;
9  using Platform.Setters;
10 using Platform.Converters;
11 using Platform.Numbers;
12 using Platform.Data.Exceptions;
13 using Platform.Data.Doublets.Decorators;
14 using Platform.Delegates;
15
16 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18 namespace Platform.Data.Doublets
19 {
20     /// <summary>
21     /// <para>
22     /// Represents the links extensions.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     public static class ILinksExtensions
27     {

```

```

28     /// <summary>
29     /// <para>
30     /// Runs the random creations using the specified links.
31     /// </para>
32     /// <para></para>
33     /// </summary>
34     /// <typeparam name="TLink">
35     /// <para>The link.</para>
36     /// <para></para>
37     /// </typeparam>
38     /// <param name="links">
39     /// <para>The links.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="amountOfCreations">
43     /// <para>The amount of creations.</para>
44     /// <para></para>
45     /// </param>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public static void RunRandomCreations<TLink>(this ILinks<TLink> links, ulong
    ↪ amountOfCreations)
48     {
49         var random = RandomHelpers.Default;
50         var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
51         var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
52         for (var i = 0UL; i < amountOfCreations; i++)
53         {
54             var linksAddressRange = new Range<ulong>(0,
    ↪ addressToUInt64Converter.Convert(links.Count()));
55             var source =
    ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
56             var target =
    ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
57             links.GetOrCreate(source, target);
58         }
59     }
60
61     /// <summary>
62     /// <para>
63     /// Runs the random searches using the specified links.
64     /// </para>
65     /// <para></para>
66     /// </summary>
67     /// <typeparam name="TLink">
68     /// <para>The link.</para>
69     /// <para></para>
70     /// </typeparam>
71     /// <param name="links">
72     /// <para>The links.</para>
73     /// <para></para>
74     /// </param>
75     /// <param name="amountOfSearches">
76     /// <para>The amount of searches.</para>
77     /// <para></para>
78     /// </param>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public static void RunRandomSearches<TLink>(this ILinks<TLink> links, ulong
    ↪ amountOfSearches)
81     {
82         var random = RandomHelpers.Default;
83         var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
84         var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
85         for (var i = 0UL; i < amountOfSearches; i++)
86         {
87             var linksAddressRange = new Range<ulong>(0,
    ↪ addressToUInt64Converter.Convert(links.Count()));
88             var source =
    ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
89             var target =
    ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
90             links.SearchOrDefault(source, target);
91         }
92     }
93
94     /// <summary>
95     /// <para>
96     /// Runs the random deletions using the specified links.
97     /// </para>

```



```

98     /// <para></para>
99     /// </summary>
100    /// <typeparam name="TLink">
101    /// <para>The link.</para>
102    /// <para></para>
103    /// </typeparam>
104    /// <param name="links">
105    /// <para>The links.</para>
106    /// <para></para>
107    /// </param>
108    /// <param name="amountOfDeletions">
109    /// <para>The amount of deletions.</para>
110    /// <para></para>
111    /// </param>
112    [MethodImpl(MethodImplOptions.AggressiveInlining)]
113    public static void RunRandomDeletions<TLink>(this ILinks<TLink> links, ulong
        ↳ amountOfDeletions)
114    {
115        var random = RandomHelpers.Default;
116        var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
117        var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
118        var linksCount = addressToUInt64Converter.Convert(links.Count());
119        var min = amountOfDeletions > linksCount ? 0UL : linksCount - amountOfDeletions;
120        for (var i = 0UL; i < amountOfDeletions; i++)
121        {
122            linksCount = addressToUInt64Converter.Convert(links.Count());
123            if (linksCount <= min)
124            {
125                break;
126            }
127            var linksAddressRange = new Range<ulong>(min, linksCount);
128            var link =
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="TLink">
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 TLink Delete<TLink>(this ILinks<TLink> links, TLink linkToDelete,
        ↳ WriteHandler<TLink>? handler)
153    {
154        if (links.Exists(linkToDelete))
155        {
156            links.EnforceResetValues(linkToDelete, handler);
157        }
158        return links.Delete(new LinkAddress<TLink>(linkToDelete), handler);
159    }
160
161    /// <remarks>
162    /// TODO: Возможно есть очень простой способ это сделать.
163    /// (Например просто удалить файл, или изменить его размер таким образом,
164    /// чтобы удалился весь контент)
165    /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
166    /// </remarks>
167    [MethodImpl(MethodImplOptions.AggressiveInlining)]
168    public static void DeleteAll<TLink>(this ILinks<TLink> links)
169    {
170        var equalityComparer = EqualityComparer<TLink>.Default;
171        var comparer = Comparer<TLink>.Default;

```

```

172     for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
173         ↪ Arithmetic.Decrement(i))
174     {
175         links.Delete(i);
176         if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
177         {
178             i = links.Count();
179         }
180     }
181 }
182
183 /// <summary>
184 /// <para>
185 /// Firsts the links.
186 /// </para>
187 /// <para></para>
188 /// </summary>
189 /// <typeparam name="TLink">
190 /// <para>The link.</para>
191 /// <para></para>
192 /// </typeparam>
193 /// <param name="links">
194 /// <para>The links.</para>
195 /// <para></para>
196 /// </param>
197 /// <exception cref="InvalidOperationException">
198 /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
199 /// <para></para>
200 /// </exception>
201 /// <exception cref="InvalidOperationException">
202 /// <para>В хранилище нет связей.</para>
203 /// <para></para>
204 /// </exception>
205 /// <returns>
206 /// <para>The first link.</para>
207 /// <para></para>
208 /// </returns>
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 public static TLink First<TLink>(this ILinks<TLink> links)
211 {
212     TLink firstLink = default;
213     var equalityComparer = EqualityComparer<TLink>.Default;
214     if (equalityComparer.Equals(links.Count(), default))
215     {
216         throw new InvalidOperationException("В хранилище нет связей.");
217     }
218     links.Each(links.Constants.Any, links.Constants.Any, link =>
219     {
220         firstLink = link[links.Constants.IndexPart];
221         return links.Constants.Break;
222     });
223     if (equalityComparer.Equals(firstLink, default))
224     {
225         throw new InvalidOperationException("В процессе поиска по хранилищу не было
226             ↪ найдено связей.");
227     }
228     return firstLink;
229 }
230
231 /// <summary>
232 /// <para>
233 /// Singles the or default using the specified links.
234 /// </para>
235 /// <para></para>
236 /// </summary>
237 /// <typeparam name="TLink">
238 /// <para>The link.</para>
239 /// <para></para>
240 /// </typeparam>
241 /// <param name="links">
242 /// <para>The links.</para>
243 /// <para></para>
244 /// </param>
245 /// <param name="query">
246 /// <para>The query.</para>
247 /// <para></para>
248 /// </param>
249 /// <returns>

```

```

248 /// <para>The result.</para>
249 /// <para></para>
250 /// </returns>
251 [MethodImpl(MethodImplOptions.AggressiveInlining)]
252 public static IList<TLink>? SingleOrDefault<TLink>(this ILinks<TLink> links,
253     ↪ IList<TLink>? query)
254 {
255     IList<TLink>? 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     TLink linkHandler(IList<TLink>? link)
264     {
265         if (count == 0)
266         {
267             result = link;
268             count++;
269             return @continue;
270         }
271         else
272         {
273             result = null;
274             return @break;
275         }
276     }
277 }
278 #region Paths
279
280 /// <remarks>
281 /// TODO: Как так? Как то что ниже может быть корректно?
282 /// Скорее всего практически не применимо
283 /// Предполагалось, что можно было конвертировать формируемый в проходе через
284     ↪ SequenceWalker
285 /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
286 /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
287 /// </remarks>
288 [MethodImpl(MethodImplOptions.AggressiveInlining)]
289 public static bool CheckPathExistance<TLink>(this ILinks<TLink> links, params TLink[]
290     ↪ path)
291 {
292     var current = path[0];
293     //EnsureLinkExists(current, "path");
294     if (!links.Exists(current))
295     {
296         return false;
297     }
298     var equalityComparer = EqualityComparer<TLink>.Default;
299     var constants = links.Constants;
300     for (var i = 1; i < path.Length; i++)
301     {
302         var next = path[i];
303         var values = links.GetLink(current);
304         var source = values[constants.SourcePart];
305         var target = values[constants.TargetPart];
306         if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
307             ↪ next))
308         {
309             //throw new InvalidOperationException(string.Format("Невозможно выбрать
310             ↪ путь, так как и Source и Target совпадают с элементом пути {0}.", next));
311             return false;
312         }
313         if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
314             ↪ target))
315         {
316             //throw new InvalidOperationException(string.Format("Невозможно продолжить
317             ↪ путь через элемент пути {0}", next));
318             return false;
319         }
320         current = next;
321     }
322     return true;
323 }
324
325 /// <remarks>

```

```

320  /// Может потребовать дополнительного стека для PathElement's при использовании
321  ↪ SequenceWalker.
322  /// </remarks>
323  [MethodImpl(MethodImplOptions.AggressiveInlining)]
324  public static TLink GetByKeys<TLink>(this ILinks<TLink> links, TLink root, params int[]
325  ↪ path)
326  {
327      links.EnsureLinkExists(root, "root");
328      var currentLink = root;
329      for (var i = 0; i < path.Length; i++)
330      {
331          currentLink = links.GetLink(currentLink)[path[i]];
332      }
333      return currentLink;
334  }
335  /// <summary>
336  /// <para>
337  /// Gets the square matrix sequence element by index using the specified links.
338  /// </para>
339  /// <para></para>
340  /// </summary>
341  /// <typeparam name="TLink">
342  /// <para>The link.</para>
343  /// <para></para>
344  /// </typeparam>
345  /// <param name="links">
346  /// <para>The links.</para>
347  /// <para></para>
348  /// </param>
349  /// <param name="root">
350  /// <para>The root.</para>
351  /// <para></para>
352  /// </param>
353  /// <param name="size">
354  /// <para>The size.</para>
355  /// <para></para>
356  /// </param>
357  /// <param name="index">
358  /// <para>The index.</para>
359  /// <para></para>
360  /// </param>
361  /// <exception cref="ArgumentOutOfRangeException">
362  /// <para>Sequences with sizes other than powers of two are not supported.</para>
363  /// <para></para>
364  /// </exception>
365  /// <returns>
366  /// <para>The current link.</para>
367  /// <para></para>
368  /// </returns>
369  [MethodImpl(MethodImplOptions.AggressiveInlining)]
370  public static TLink GetSquareMatrixSequenceElementByIndex<TLink>(this ILinks<TLink>
371  ↪ links, TLink root, ulong size, ulong index)
372  {
373      var constants = links.Constants;
374      var source = constants.SourcePart;
375      var target = constants.TargetPart;
376      if (!Platform.Numbers.Math.IsPowerOfTwo(size))
377      {
378          throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
379          ↪ than powers of two are not supported.");
380      }
381      var path = new BitArray(BitConverter.GetBytes(index));
382      var length = Bit.GetLowestPosition(size);
383      links.EnsureLinkExists(root, "root");
384      var currentLink = root;
385      for (var i = length - 1; i >= 0; i--)
386      {
387          currentLink = links.GetLink(currentLink)[path[i] ? target : source];
388      }
389      return currentLink;
390  }
391  #endregion
392  /// <summary>
393  /// Возвращает индекс указанной связи.
394  /// </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 TLink GetIndex<TLink>(this ILinks<TLink> links, IList<TLink>? 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 TLink GetSource<TLink>(this ILinks<TLink> links, TLink link) =>
    ↳ links.GetLink(link)[links.Constants.SourcePart];

408
409 /// <summary>
410 /// Возвращает индекс начальной (Source) связи для указанной связи.
411 /// </summary>
412 /// <param name="links">Хранилище связей.</param>
413 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
414 /// <returns>Индекс начальной связи для указанной связи.</returns>
415 [MethodImpl(MethodImplOptions.AggressiveInlining)]
416 public static TLink GetSource<TLink>(this ILinks<TLink> links, IList<TLink>? link) =>
    ↳ link[links.Constants.SourcePart];

417
418 /// <summary>
419 /// Возвращает индекс конечной (Target) связи для указанной связи.
420 /// </summary>
421 /// <param name="links">Хранилище связей.</param>
422 /// <param name="link">Индекс связи.</param>
423 /// <returns>Индекс конечной связи для указанной связи.</returns>
424 [MethodImpl(MethodImplOptions.AggressiveInlining)]
425 public static TLink GetTarget<TLink>(this ILinks<TLink> links, TLink link) =>
    ↳ links.GetLink(link)[links.Constants.TargetPart];

426
427 /// <summary>
428 /// Возвращает индекс конечной (Target) связи для указанной связи.
429 /// </summary>
430 /// <param name="links">Хранилище связей.</param>
431 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
432 /// <returns>Индекс конечной связи для указанной связи.</returns>
433 [MethodImpl(MethodImplOptions.AggressiveInlining)]
434 public static TLink GetTarget<TLink>(this ILinks<TLink> links, IList<TLink>? link) =>
    ↳ link[links.Constants.TargetPart];

435
436 /// <summary>
437 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
438 /// </summary>
439 /// <param name="links">Хранилище связей.</param>
440 /// <param name="handler">Обработчик каждой подходящей связи.</param>
441 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
    ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Any -
    ↳ отсутствие ограничения, 1..∞ конкретный адрес связи.</param>
442 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    ↳ случае.</returns>
443 [MethodImpl(MethodImplOptions.AggressiveInlining)]
444 public static bool Each<TLink>(this ILinks<TLink> links, ReadHandler<TLink>? handler,
    ↳ params TLink[] restriction)
445     => EqualityComparer<TLink>.Default.Equals(links.Each(restriction, handler),
    ↳ links.Constants.Continue);

446
447 public static bool Each<TLink>(this ILinks<TLink> links, Func<TLink, bool> handler,
    ↳ TLink source, TLink target) => links.Each(source, target, handler);

448
449 /// <summary>
450 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    ↳ (handler) для каждой подходящей связи.
451 /// </summary>
452 /// <param name="links">Хранилище связей.</param>
453

```

```

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

```

```

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

```

```

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="TLink">
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{TLink}">
611 /// <para></para>
612 /// <para></para>
613 /// </exception>
614 [MethodImpl(MethodImplOptions.AggressiveInlining)]
615 public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links, TLink
    ↳ reference, string argumentName)
616 {
617     if (links.Constants.IsInternalReference(reference) && !links.Exists(reference))
618     {
619         throw new ArgumentLinkDoesNotExistsException<TLink>(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="TLink">
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<TLink>(this ILinks<TLink> links,
    ↳ IList<TLink>? 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="TLink">
661    /// <para>The link.</para>
662    /// <para></para>
663    /// </typeparam>
664    /// <param name="links">
665    /// <para>The links.</para>
666    /// <para></para>
667    /// </param>
668    /// <param name="restriction">
669    /// <para>The restriction.</para>
670    /// <para></para>
671    /// </param>
672    /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
673    /// <para>sequence[{i}]</para>
674    /// <para></para>
675    /// </exception>
676    [MethodImpl(MethodImplOptions.AggressiveInlining)]
677    public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links,
678    ↪ IList<TLink>? restriction)
679    {
680        var equalityComparer = EqualityComparer<TLink>.Default;
681        var any = links.Constants.Any;
682        for (var i = 0; i < restriction.Count; i++)
683        {
684            if (!equalityComparer.Equals(restriction[i], any) &&
685            ↪ !links.Exists(restriction[i]))
686            {
687                throw new ArgumentLinkDoesNotExistsException<TLink>(restriction[i],
688                ↪ $"sequence[{i}]");
689            }
690        }
691    }
692
693    /// <summary>
694    /// <para>
695    /// Ensures the link is any or exists using the specified links.
696    /// </para>
697    /// <para></para>
698    /// </summary>
699    /// <typeparam name="TLink">
700    /// <para>The link.</para>
701    /// <para></para>
702    /// </typeparam>
703    /// <param name="links">
704    /// <para>The links.</para>
705    /// <para></para>
706    /// </param>
707    /// <param name="link">
708    /// <para>The link.</para>
709    /// <para></para>
710    /// </param>
711    /// <param name="argumentName">
712    /// <para>The argument name.</para>
713    /// <para></para>
714    /// </param>
715    /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
716    /// <para></para>
717    /// <para></para>
718    /// </exception>
719    [MethodImpl(MethodImplOptions.AggressiveInlining)]
720    public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links, TLink link,
721    ↪ string argumentName)
722    {
723        var equalityComparer = EqualityComparer<TLink>.Default;
724        if (!equalityComparer.Equals(link, links.Constants.Any) && !links.Exists(link))
725        {
726            throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
727        }
728    }
729
730    /// <summary>
731    /// <para>
732    /// Ensures the link is itself or exists using the specified links.
733    /// </para>
734    /// <para></para>
735    /// </summary>
736    /// <typeparam name="TLink">
737    /// <para>The link.</para>

```

```

734 /// <para></para>
735 /// </typeparam>
736 /// <param name="links">
737 /// <para>The links.</para>
738 /// <para></para>
739 /// </param>
740 /// <param name="link">
741 /// <para>The link.</para>
742 /// <para></para>
743 /// </param>
744 /// <param name="argumentName">
745 /// <para>The argument name.</para>
746 /// <para></para>
747 /// </param>
748 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
749 /// <para></para>
750 /// <para></para>
751 /// </exception>
752 [MethodImpl(MethodImplOptions.AggressiveInlining)]
753 public static void EnsureLinkIsItselfOrExists<TLink>(this ILinks<TLink> links, TLink
    ↪ link, string argumentName)
754 {
755     var equalityComparer = EqualityComparer<TLink>.Default;
756     if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))
757     {
758         throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
759     }
760 }
761
762 /// <param name="links">Хранилище связей.</param>
763 [MethodImpl(MethodImplOptions.AggressiveInlining)]
764 public static void EnsureDoesNotExists<TLink>(this ILinks<TLink> links, TLink source,
    ↪ TLink target)
765 {
766     if (links.Exists(source, target))
767     {
768         throw new LinkWithSameValueAlreadyExistsException();
769     }
770 }
771
772 /// <param name="links">Хранилище связей.</param>
773 [MethodImpl(MethodImplOptions.AggressiveInlining)]
774 public static void EnsureNoUsages<TLink>(this ILinks<TLink> links, TLink link)
775 {
776     if (links.HasUsages(link))
777     {
778         throw new ArgumentLinkHasDependenciesException<TLink>(link);
779     }
780 }
781
782 /// <param name="links">Хранилище связей.</param>
783 [MethodImpl(MethodImplOptions.AggressiveInlining)]
784 public static void EnsureCreated<TLink>(this ILinks<TLink> links, params TLink[]
    ↪ addresses) => links.EnsureCreated(links.Create, addresses);
785
786 /// <param name="links">Хранилище связей.</param>
787 [MethodImpl(MethodImplOptions.AggressiveInlining)]
788 public static void EnsurePointsCreated<TLink>(this ILinks<TLink> links, params TLink[]
    ↪ addresses) => links.EnsureCreated(links.CreatePoint, addresses);
789
790 /// <param name="links">Хранилище связей.</param>
791 [MethodImpl(MethodImplOptions.AggressiveInlining)]
792 public static void EnsureCreated<TLink>(this ILinks<TLink> links, Func<TLink> creator,
    ↪ params TLink[] addresses)
793 {
794     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
795     var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
796     var nonExistentAddresses = new HashSet<TLink>(addresses.Where(x =>
    ↪ !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<TLink>();
802         var equalityComparer = EqualityComparer<TLink>.Default;

```

```

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

```

```

873     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
874     links.CreatePoint(setter.SetFirstFromSecondListAndReturnTrue);
875     return setter.Result;
876 }
877
878 /// <param name="links">Хранилище связей.</param>
879 [MethodImpl(MethodImplOptions.AggressiveInlining)]
880 public static TLink CreatePoint<TLink>(this ILinks<TLink> links, WriteHandler<TLink>?
    ↪ handler)
881 {
882     var constants = links.Constants;
883     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
    ↪ handler);
884     TLink link = default;
885     TLink HandlerWrapper(IList<TLink>? before, IList<TLink>? after)
886     {
887         link = after[constants.IndexPart];
888         return handlerState.Handler != null ? handlerState.Handler(before, after) :
    ↪ constants.Continue;
889     }
890     handlerState.Apply(links.Create(null, HandlerWrapper));
891     handlerState.Apply(links.Update(link, link, link, HandlerWrapper));
892     return handlerState.Result;
893 }
894
895 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↪ target)
896 {
897     var constants = links.Constants;
898     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
899     links.CreateAndUpdate(source, target, setter.SetFirstFromSecondListAndReturnTrue);
900     return setter.Result;
901 }
902
903
904 /// <param name="links">Хранилище связей.</param>
905 [MethodImpl(MethodImplOptions.AggressiveInlining)]
906 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↪ target, WriteHandler<TLink>? handler)
907 {
908     var constants = links.Constants;
909     TLink createdLink = default;
910     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
    ↪ handler);
911     handlerState.Apply(links.Create(null, (before, after) =>
912     {
913         createdLink = links.GetIndex(after);
914         return handlerState.Handler != null ? handlerState.Handler(before, after) :
    ↪ constants.Continue;
915     })));
916     handlerState.Apply(links.Update(createdLink, source, target, handler));
917     return handlerState.Result;
918 }
919
920 /// <summary>
921 /// Обновляет связь с указанными началом (Source) и концом (Target)
922 /// на связь с указанными началом (NewSource) и концом (NewTarget).
923 /// </summary>
924 /// <param name="links">Хранилище связей.</param>
925 /// <param name="link">Индекс обновляемой связи.</param>
926 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
    ↪ выполняется обновление.</param>
927 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
    ↪ выполняется обновление.</param>
928 /// <returns>Индекс обновлённой связи.</returns>
929 [MethodImpl(MethodImplOptions.AggressiveInlining)]
930 public static TLink Update<TLink>(this ILinks<TLink> links, TLink link, TLink newSource,
    ↪ TLink newTarget) => links.Update(new LinkAddress<TLink>(link), new Link<TLink>(link,
    ↪ newSource, newTarget));
931
932 public static TLink Update<TLink>(this ILinks<TLink> links, params TLink[] restriction)
    ↪ => links.Update(restriction, null);
933
934 public static TLink Update<TLink>(this ILinks<TLink> links, WriteHandler<TLink>?
    ↪ handler, params TLink[] restriction) => links.Update(restriction, handler);
935
936 public static TLink Update<TLink>(this ILinks<TLink> links, IList<TLink>? restriction)
937 {

```

```

938     var constants = links.Constants;
939     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
940     links.Update(restriction, setter.SetFirstFromSecondListAndReturnTrue);
941     return setter.Result;
942 }
943
944
945 /// <summary>
946 /// Обновляет связь с указанными началом (Source) и концом (Target)
947 /// на связь с указанными началом (NewSource) и концом (NewTarget).
948 /// </summary>
949 /// <param name="links">Хранилище связей.</param>
950 /// <param name="restriction">Ограничения на содержимое связей. Каждое ограничение может
951   ↳ иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту, Itself -
952   ↳ требование установить ссылку на себя, 1.. $\infty$  конкретный адрес другой связи.</param>
953 /// <returns>Индекс обновлённой связи.</returns>
954 [MethodImpl(MethodImplOptions.AggressiveInlining)]
955 public static TLink Update<TLink>(this ILinks<TLink> links, IList<TLink>? restriction,
956   ↳ WriteHandler<TLink>? handler)
957 {
958     return restriction.Count switch
959     {
960         2 => links.MergeAndDelete(restriction[0], restriction[1], handler),
961         4 => links.UpdateOrCreateOrGet(restriction[0], restriction[1], restriction[2],
962           ↳ restriction[3], handler),
963         _ => links.Update(restriction[0], restriction[1], restriction[2], handler)
964     };
965 }
966
967 public static TLink Update<TLink>(this ILinks<TLink> links, TLink link, TLink newSource,
968   ↳ TLink newTarget, WriteHandler<TLink>? handler) => links.Update(new
969   ↳ LinkAddress<TLink>(link), new Link<TLink>(link, newSource, newTarget), handler);
970
971 /// <summary>
972 /// <para>
973 /// Resolves the constant as self reference using the specified links.
974 /// </para>
975 /// <para></para>
976 /// </summary>
977 /// <typeparam name="TLink">
978 /// <para>The link.</para>
979 /// <para></para>
980 /// </typeparam>
981 /// <param name="links">
982 /// <para>The links.</para>
983 /// <para></para>
984 /// </param>
985 /// <param name="constant">
986 /// <para>The constant.</para>
987 /// <para></para>
988 /// </param>
989 /// <param name="restriction">
990 /// <para>The restriction.</para>
991 /// <para></para>
992 /// </param>
993 /// <param name="substitution">
994 /// <para>The substitution.</para>
995 /// <para></para>
996 /// </param>
997 /// <returns>
998 /// <para>A list of t link</para>
999 /// <para></para>
1000 /// </returns>
1001 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1002 public static IList<TLink>? ResolveConstantAsSelfReference<TLink>(this ILinks<TLink>
1003   ↳ links, TLink constant, IList<TLink>? restriction, IList<TLink>? substitution)
1004 {
1005     var equalityComparer = EqualityComparer<TLink>.Default;
1006     var constants = links.Constants;
1007     var restrictionIndex = restriction[constants.IndexPart];
1008     var substitutionIndex = substitution[constants.IndexPart];
1009     if (equalityComparer.Equals(substitutionIndex, default))
1010     {
1011         substitutionIndex = restrictionIndex;
1012     }
1013     var source = substitution[constants.SourcePart];
1014     var target = substitution[constants.TargetPart];
1015     source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;

```

```

1009         target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
1010         return new Link<TLink>(substitutionIndex, source, target);
1011     }
1012
1013     /// <summary>
1014     /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
1015     /// с указанными Source (началом) и Target (концом).
1016     /// </summary>
1017     /// <param name="links">Хранилище связей.</param>
1018     /// <param name="source">Индекс связи, которая является началом на создаваемой
1019     /// связи.</param>
1020     /// <param name="target">Индекс связи, которая является концом для создаваемой
1021     /// связи.</param>
1022     /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
1023     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1024     public static TLink GetOrCreate<TLink>(this ILinks<TLink> links, TLink source, TLink
1025     → target)
1026     {
1027         var link = links.SearchOrDefault(source, target);
1028         if (EqualityComparer<TLink>.Default.Equals(link, default))
1029         {
1030             link = links.CreateAndUpdate(source, target);
1031         }
1032         return link;
1033     }
1034
1035     public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
1036     → TLink target, TLink newSource, TLink newTarget)
1037     {
1038         var constants = links.Constants;
1039         var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break);
1040         links.UpdateOrCreateOrGet(source, target, newSource, newTarget,
1041         → setter.SetFirstFromSecondListAndReturnTrue);
1042         return setter.Result;
1043     }
1044
1045     /// <summary>
1046     /// Обновляет связь с указанными началом (Source) и концом (Target)
1047     /// на связь с указанными началом (NewSource) и концом (NewTarget).
1048     /// </summary>
1049     /// <param name="links">Хранилище связей.</param>
1050     /// <param name="source">Индекс связи, которая является началом обновляемой
1051     /// связи.</param>
1052     /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
1053     /// <param name="newSource">Индекс связи, которая является началом связи, на которую
1054     /// выполняется обновление.</param>
1055     /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
1056     /// выполняется обновление.</param>
1057     /// <returns>Индекс обновлённой связи.</returns>
1058     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1059     public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
1060     → TLink target, TLink newSource, TLink newTarget, WriteHandler<TLink>? handler)
1061     {
1062         var equalityComparer = EqualityComparer<TLink>.Default;
1063         var link = links.SearchOrDefault(source, target);
1064         if (equalityComparer.Equals(link, default))
1065         {
1066             return links.CreateAndUpdate(newSource, newTarget, handler);
1067         }
1068         if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
1069         → target))
1070         {
1071             var linkStruct = new Link<TLink>(link, source, target);
1072             return link;
1073         }
1074         return links.Update(link, newSource, newTarget, handler);
1075     }
1076
1077     /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1078     /// <param name="links">Хранилище связей.</param>
1079     /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
1080     /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
1081     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1082     public static TLink DeleteIfExists<TLink>(this ILinks<TLink> links, TLink source, TLink
1083     → target)
1084     {
1085         var link = links.SearchOrDefault(source, target);
1086     }

```

```

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

```

```

1144         links.ForEach(queryResultFiller.AddFirstAndReturnConstant, query);
1145         foreach (var link in queryResult)
1146         {
1147             links.Delete(link);
1148         }
1149     }
1150
1151     // TODO: Move to Platform.Data
1152     /// <summary>
1153     /// <para>
1154     /// Determines whether are values reset.
1155     /// </para>
1156     /// <para></para>
1157     /// </summary>
1158     /// <typeparam name="TLink">
1159     /// <para>The link.</para>
1160     /// <para></para>
1161     /// </typeparam>
1162     /// <param name="links">
1163     /// <para>The links.</para>
1164     /// <para></para>
1165     /// </param>
1166     /// <param name="linkIndex">
1167     /// <para>The link index.</para>
1168     /// <para></para>
1169     /// </param>
1170     /// <returns>
1171     /// <para>The bool</para>
1172     /// <para></para>
1173     /// </returns>
1174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1175     public static bool AreValuesReset<TLink>(this ILinks<TLink> links, TLink linkIndex)
1176     {
1177         var nullConstant = links.Constants.Null;
1178         var equalityComparer = EqualityComparer<TLink>.Default;
1179         var link = links.GetLink(linkIndex);
1180         for (int i = 1; i < link.Count; i++)
1181         {
1182             if (!equalityComparer.Equals(link[i], nullConstant))
1183             {
1184                 return false;
1185             }
1186         }
1187         return true;
1188     }
1189
1190     public static void ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex) =>
1191     ↪ links.ResetValues(linkIndex, null);
1192
1193     // TODO: Create a universal version of this method in Platform.Data (with using of for
1194     ↪ loop)
1195     /// <summary>
1196     /// <para>
1197     /// Resets the values using the specified links.
1198     /// </para>
1199     /// <para></para>
1200     /// </summary>
1201     /// <typeparam name="TLink">
1202     /// <para>The link.</para>
1203     /// <para></para>
1204     /// </typeparam>
1205     /// <param name="links">
1206     /// <para>The links.</para>
1207     /// <para></para>
1208     /// </param>
1209     /// <param name="linkIndex">
1210     /// <para>The link index.</para>
1211     /// <para></para>
1212     /// </param>
1213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1214     public static TLink ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex,
1215     ↪ WriteHandler<TLink>? handler)
1216     {
1217         var nullConstant = links.Constants.Null;
1218         var updateRequest = new Link<TLink>(linkIndex, nullConstant, nullConstant);
1219         return links.Update(updateRequest, handler);
1220     }
1221
1222     }
1223
1224     }
1225
1226     }
1227
1228     }

```



```

1219 public static void EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1220     => 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="TLink">
1231 /// <para>The link.</para>
1232 /// </typeparam>
1233 /// <param name="links">
1234 /// <para>The links.</para>
1235 /// <para></para>
1236 /// </param>
1237 /// <param name="linkIndex">
1238 /// <para>The link index.</para>
1239 /// <para></para>
1240 /// </param>
1241 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1242 public static TLink EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex,
1243     => WriteHandler<TLink>? handler)
1244 {
1245     if (!links.AreValuesReset(linkIndex))
1246     {
1247         return links.ResetValues(linkIndex, handler);
1248     }
1249     return links.Constants.Continue;
1250 }
1251
1252 public static void MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1253     => TLink newLinkIndex) => links.MergeUsages(oldLinkIndex, newLinkIndex, null);
1254
1255 /// <summary>
1256 /// Merging two usages graphs, all children of old link moved to be children of new link
1257     => or deleted.
1258 /// </summary>
1259 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1260 public static TLink MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1261     => TLink newLinkIndex, WriteHandler<TLink>? handler)
1262 {
1263     var equalityComparer = EqualityComparer<TLink>.Default;
1264     if (equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1265     {
1266         return newLinkIndex;
1267     }
1268     var constants = links.Constants;
1269     var usagesAsSource = links.All(new Link<TLink>(constants.Any, oldLinkIndex,
1270     => constants.Any));
1271     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
1272     => handler);
1273     for (var i = 0; i < usagesAsSource.Count; i++)
1274     {
1275         var usageAsSource = usagesAsSource[i];
1276         if (equalityComparer.Equals(usageAsSource[constants.IndexPart], oldLinkIndex))
1277         {
1278             continue;
1279         }
1280         var restriction = new LinkAddress<TLink>(usageAsSource[constants.IndexPart]);
1281         var substitution = new Link<TLink>(newLinkIndex,
1282     => usageAsSource[constants.TargetPart]);
1283         handlerState.Apply(links.Update(restriction, substitution,
1284     => handlerState.Handler));
1285     }
1286     var usagesAsTarget = links.All(new Link<TLink>(constants.Any, constants.Any,
1287     => oldLinkIndex));
1288     for (var i = 0; i < usagesAsTarget.Count; i++)
1289     {
1290         var usageAsTarget = usagesAsTarget[i];
1291         if (equalityComparer.Equals(usageAsTarget[constants.IndexPart], oldLinkIndex))
1292         {
1293             continue;
1294         }
1295     }

```

```

1285     }
1286     var restriction = links.GetLink(usageAsTarget[constants.IndexPart]);
1287     var substitution = new Link<TLink>(usageAsTarget[constants.TargetPart],
1288         ↪ newLinkIndex);
1289     handlerState.Apply(links.Update(restriction, substitution,
1290         ↪ handlerState.Handler));
1291     }
1292     return handlerState.Result;
1293 }
1294
1295 public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1296     ↪ TLink newLinkIndex)
1297 {
1298     var equalityComparer = EqualityComparer<TLink>.Default;
1299     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1300     {
1301         links.MergeUsages(oldLinkIndex, newLinkIndex);
1302         links.Delete(oldLinkIndex);
1303     }
1304     return newLinkIndex;
1305 }
1306
1307 /// <summary>
1308 /// Replace one link with another (replaced link is deleted, children are updated or
1309 ↪ deleted).
1310 /// </summary>
1311 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1312 public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1313     ↪ TLink newLinkIndex, WriteHandler<TLink>? handler)
1314 {
1315     var equalityComparer = EqualityComparer<TLink>.Default;
1316     var constants = links.Constants;
1317     WriteHandlerState<TLink> handlerState = new(constants.Continue, constants.Break,
1318         ↪ handler);
1319     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1320     {
1321         handlerState.Apply(links.MergeUsages(oldLinkIndex, newLinkIndex,
1322             ↪ handlerState.Handler));
1323         handlerState.Apply(links.Delete(oldLinkIndex, handlerState.Handler));
1324     }
1325     return handlerState.Result;
1326 }
1327
1328 /// <summary>
1329 /// <para>
1330 /// Decorates the with automatic uniqueness and usages resolution using the specified
1331 ↪ links.
1332 /// </para>
1333 /// <para></para>
1334 /// </summary>
1335 /// <typeparam name="TLink">
1336 /// <para>The link.</para>
1337 /// <para></para>
1338 /// </typeparam>
1339 /// <param name="links">
1340 /// <para>The links.</para>
1341 /// <para></para>
1342 /// </param>
1343 /// <returns>
1344 /// <para>The links.</para>
1345 /// <para></para>
1346 /// </returns>
1347 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1348 public static ILinks<TLink>
1349     ↪ DecorateWithAutomaticUniquenessAndUsagesResolution<TLink>(this ILinks<TLink> links)
1350 {
1351     links = new LinksCascadeUsagesResolver<TLink>(links);
1352     links = new NonNullContentsLinkDeletionResolver<TLink>(links);
1353     links = new LinksCascadeUniquenessAndUsagesResolver<TLink>(links);
1354     return links;
1355 }
1356
1357 /// <summary>
1358 /// <para>
1359 /// Formats the links.
1360 /// </para>
1361 /// <para></para>
1362 /// </summary>

```

```

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

```

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

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets
4  {
5      /// <summary>
6      /// <para>
7      /// Defines the synchronized links.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     /// <seealso cref="ISynchronizedLinks{TLink, ILinks{TLink}, LinksConstants{TLink}}"/>
12     /// <seealso cref="ILinks{TLink}"/>
13     public interface ISynchronizedLinks<TLink> : ISynchronizedLinks<TLink, ILinks<TLink>,
14         ↳ LinksConstants<TLink>>, ILinks<TLink>
15     {
16     }

```

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

```

1  using Platform.Collections.Lists;
2  using Platform.Exceptions;
3  using Platform.Ranges;
4  using Platform.Singletons;
5  using System;
6  using System.Collections;

```

```

7 using System.Collections.Generic;
8 using System.Runtime.CompilerServices;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets
13 {
14     /// <summary>
15     /// Структура описывающая уникальную связь.
16     /// </summary>
17     public struct Link<TLink> : IEquatable<Link<TLink>>, IReadOnlyList<TLink>, IList<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// The link.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         public static readonly Link<TLink> Null = new Link<TLink>();
26         private static readonly LinksConstants<TLink> _constants =
27             ↪ Default<LinksConstants<TLink>>.Instance;
28         private static readonly EqualityComparer<TLink> _equalityComparer =
29             ↪ EqualityComparer<TLink>.Default;
30         private const int Length = 3;
31
32         /// <summary>
33         /// <para>
34         /// The index.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         public readonly TLink Index;
39         /// <summary>
40         /// <para>
41         /// The source.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         public readonly TLink Source;
46         /// <summary>
47         /// <para>
48         /// The target.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         public readonly TLink Target;
53
54         /// <summary>
55         /// <para>
56         /// Initializes a new <see cref="Link"/> instance.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         /// <param name="values">
61         /// <para>A values.</para>
62         /// <para></para>
63         /// </param>
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         public Link(params TLink[] values) => SetValues(values, out Index, out Source, out
66             ↪ Target);
67
68         /// <summary>
69         /// <para>
70         /// Initializes a new <see cref="Link"/> instance.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <param name="values">
75         /// <para>A values.</para>
76         /// <para></para>
77         /// </param>
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         public Link(IList<TLink>? values) => SetValues(values, out Index, out Source, out
80             ↪ Target);
81
82         /// <summary>
83         /// <para>
84         /// Initializes a new <see cref="Link"/> instance.
85         /// </para>

```

```

82     /// <para></para>
83     /// </summary>
84     /// <param name="other">
85     /// <para>A other.</para>
86     /// <para></para>
87     /// </param>
88     /// <exception cref="NotSupportedException">
89     /// <para></para>
90     /// <para></para>
91     /// </exception>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public Link(object other)
94     {
95         if (other is Link<TLink> otherLink)
96         {
97             SetValues(ref otherLink, out Index, out Source, out Target);
98         }
99         else if (other is IList<TLink> otherList)
100        {
101            SetValues(otherList, out Index, out Source, out Target);
102        }
103        else
104        {
105            throw new NotSupportedException();
106        }
107    }
108
109    /// <summary>
110    /// <para>
111    /// Initializes a new <see cref="Link"/> instance.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="other">
116    /// <para>A other.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    public Link(ref Link<TLink> other) => SetValues(ref other, out Index, out Source, out
    ↪ 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(TLink index, TLink source, TLink target)
142    {
143        Index = index;
144        Source = source;
145        Target = target;
146    }
147    [MethodImpl(MethodImplOptions.AggressiveInlining)]
148    private static void SetValues(ref Link<TLink> other, out TLink index, out TLink source,
    ↪ out TLink 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<TLink>? values, out TLink index, out TLink source,
    ↪ out TLink target)
156    {

```

```

157     if (values == null)
158     {
159         index = default;
160         source = default;
161         target = default;
162         return;
163     }
164     switch (values.Count)
165     {
166         case 3:
167             index = values[0];
168             source = values[1];
169             target = values[2];
170             break;
171         case 2:
172             index = values[0];
173             source = values[1];
174             target = default;
175             break;
176         case 1:
177             index = values[0];
178             source = default;
179             target = default;
180             break;
181         default:
182             index = default;
183             source = default;
184             target = default;
185             break;
186     }
187 }
188
189 /// <summary>
190 /// <para>
191 /// Gets the hash code.
192 /// </para>
193 /// <para></para>
194 /// </summary>
195 /// <returns>
196 /// <para>The int</para>
197 /// <para></para>
198 /// </returns>
199 [MethodImpl(MethodImplOptions.AggressiveInlining)]
200 public override int GetHashCode() => (Index, Source, Target).GetHashCode();
201
202 /// <summary>
203 /// <para>
204 /// Determines whether this instance is null.
205 /// </para>
206 /// <para></para>
207 /// </summary>
208 /// <returns>
209 /// <para>The bool</para>
210 /// <para></para>
211 /// </returns>
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
214     && _equalityComparer.Equals(Source, _constants.Null)
215     && _equalityComparer.Equals(Target, _constants.Null);
216
217 /// <summary>
218 /// <para>
219 /// Determines whether this instance equals.
220 /// </para>
221 /// <para></para>
222 /// </summary>
223 /// <param name="other">
224 /// <para>The other.</para>
225 /// <para></para>
226 /// </param>
227 /// <returns>
228 /// <para>The bool</para>
229 /// <para></para>
230 /// </returns>
231 [MethodImpl(MethodImplOptions.AggressiveInlining)]
232 public override bool Equals(object other) => other is Link<TLink> &&
233     ↪ Equals((Link<TLink>)other);
234
235 /// <summary>

```

```

235     /// <para>
236     /// Determines whether this instance equals.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="other">
241     /// <para>The other.</para>
242     /// <para></para>
243     /// </param>
244     /// <returns>
245     /// <para>The bool</para>
246     /// <para></para>
247     /// </returns>
248     [MethodImpl(MethodImplOptions.AggressiveInlining)]
249     public bool Equals(Link<TLink> other) => _equalityComparer.Equals(Index, other.Index)
250         && _equalityComparer.Equals(Source, other.Source)
251         && _equalityComparer.Equals(Target, other.Target);
252
253     /// <summary>
254     /// <para>
255     /// Returns the string using the specified index.
256     /// </para>
257     /// <para></para>
258     /// </summary>
259     /// <param name="index">
260     /// <para>The index.</para>
261     /// <para></para>
262     /// </param>
263     /// <param name="source">
264     /// <para>The source.</para>
265     /// <para></para>
266     /// </param>
267     /// <param name="target">
268     /// <para>The target.</para>
269     /// <para></para>
270     /// </param>
271     /// <returns>
272     /// <para>The string</para>
273     /// <para></para>
274     /// </returns>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     public static string ToString(TLink index, TLink source, TLink target) => $"({index}:
    ↳ {source}->{target})";
277
278     /// <summary>
279     /// <para>
280     /// Returns the string using the specified source.
281     /// </para>
282     /// <para></para>
283     /// </summary>
284     /// <param name="source">
285     /// <para>The source.</para>
286     /// <para></para>
287     /// </param>
288     /// <param name="target">
289     /// <para>The target.</para>
290     /// <para></para>
291     /// </param>
292     /// <returns>
293     /// <para>The string</para>
294     /// <para></para>
295     /// </returns>
296     [MethodImpl(MethodImplOptions.AggressiveInlining)]
297     public static string ToString(TLink source, TLink target) => $"({source}->{target})";
298
299     [MethodImpl(MethodImplOptions.AggressiveInlining)]
300     public static implicit operator TLink[] (Link<TLink> link) => link.ToArray();
301
302     [MethodImpl(MethodImplOptions.AggressiveInlining)]
303     public static implicit operator Link<TLink> (TLink[] linkArray) => new
    ↳ Link<TLink>(linkArray);
304
305     /// <summary>
306     /// <para>
307     /// Returns the string.
308     /// </para>
309     /// <para></para>
310     /// </summary>

```

```

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

317
318     #region IList
319
320     /// <summary>
321     /// <para>
322     /// Gets the count value.
323     /// </para>
324     /// <para></para>
325     /// </summary>
326     public int Count
327     {
328         [MethodImpl(MethodImplOptions.AggressiveInlining)]
329         get => Length;
330     }
331
332     /// <summary>
333     /// <para>
334     /// Gets the is read only value.
335     /// </para>
336     /// <para></para>
337     /// </summary>
338     public bool IsReadOnly
339     {
340         [MethodImpl(MethodImplOptions.AggressiveInlining)]
341         get => true;
342     }
343
344     /// <summary>
345     /// <para>
346     /// The not supported exception.
347     /// </para>
348     /// <para></para>
349     /// </summary>
350     public TLink this[int index]
351     {
352         [MethodImpl(MethodImplOptions.AggressiveInlining)]
353         get
354         {
355             Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
        ↳ nameof(index));
356             if (index == _constants.IndexPart)
357             {
358                 return Index;
359             }
360             if (index == _constants.SourcePart)
361             {
362                 return Source;
363             }
364             if (index == _constants.TargetPart)
365             {
366                 return Target;
367             }
368             throw new NotSupportedException(); // Impossible path due to
        ↳ Ensure.ArgumentInRange
369         }
370         [MethodImpl(MethodImplOptions.AggressiveInlining)]
371         set => throw new NotSupportedException();
372     }
373
374     /// <summary>
375     /// <para>
376     /// Gets the enumerator.
377     /// </para>
378     /// <para></para>
379     /// </summary>
380     /// <returns>
381     /// <para>The enumerator</para>
382     /// <para></para>
383     /// </returns>
384     [MethodImpl(MethodImplOptions.AggressiveInlining)]
385     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
386

```



```

387     /// <summary>
388     /// <para>
389     /// Gets the enumerator.
390     /// </para>
391     /// <para></para>
392     /// </summary>
393     /// <returns>
394     /// <para>An enumerator of t link</para>
395     /// <para></para>
396     /// </returns>
397     [MethodImpl(MethodImplOptions.AggressiveInlining)]
398     public IEnumerator<TLink> GetEnumerator()
399     {
400         yield return Index;
401         yield return Source;
402         yield return Target;
403     }
404
405     /// <summary>
406     /// <para>
407     /// Adds the item.
408     /// </para>
409     /// <para></para>
410     /// </summary>
411     /// <param name="item">
412     /// <para>The item.</para>
413     /// <para></para>
414     /// </param>
415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     public void Add(TLink item) => throw new NotSupportedException();
417
418     /// <summary>
419     /// <para>
420     /// Clears this instance.
421     /// </para>
422     /// <para></para>
423     /// </summary>
424     [MethodImpl(MethodImplOptions.AggressiveInlining)]
425     public void Clear() => throw new NotSupportedException();
426
427     /// <summary>
428     /// <para>
429     /// Determines whether this instance contains.
430     /// </para>
431     /// <para></para>
432     /// </summary>
433     /// <param name="item">
434     /// <para>The item.</para>
435     /// <para></para>
436     /// </param>
437     /// <returns>
438     /// <para>The bool</para>
439     /// <para></para>
440     /// </returns>
441     [MethodImpl(MethodImplOptions.AggressiveInlining)]
442     public bool Contains(TLink item) => IndexOf(item) >= 0;
443
444     /// <summary>
445     /// <para>
446     /// Copies the to using the specified array.
447     /// </para>
448     /// <para></para>
449     /// </summary>
450     /// <param name="array">
451     /// <para>The array.</para>
452     /// <para></para>
453     /// </param>
454     /// <param name="arrayIndex">
455     /// <para>The array index.</para>
456     /// <para></para>
457     /// </param>
458     /// <exception cref="InvalidOperationException">
459     /// <para></para>
460     /// <para></para>
461     /// </exception>
462     [MethodImpl(MethodImplOptions.AggressiveInlining)]
463     public void CopyTo(TLink[] array, int arrayIndex)
464     {

```

```

465     Ensure.OnDebug.ArgumentNotNull(array, nameof(array));
466     Ensure.OnDebug.ArgumentInRange(arrayIndex, new Range<int>(0, array.Length - 1),
    ↪     nameof(arrayIndex));
467     if (arrayIndex + Length > array.Length)
468     {
469         throw new InvalidOperationException();
470     }
471     array[arrayIndex++] = Index;
472     array[arrayIndex++] = Source;
473     array[arrayIndex] = Target;
474 }
475
476 /// <summary>
477 /// <para>
478 /// Determines whether this instance remove.
479 /// </para>
480 /// <para></para>
481 /// </summary>
482 /// <param name="item">
483 /// <para>The item.</para>
484 /// <para></para>
485 /// </param>
486 /// <returns>
487 /// <para>The bool</para>
488 /// <para></para>
489 /// </returns>
490 [MethodImpl(MethodImplOptions.AggressiveInlining)]
491 public bool Remove(TLink item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
492
493 /// <summary>
494 /// <para>
495 /// Indexes the of using the specified item.
496 /// </para>
497 /// <para></para>
498 /// </summary>
499 /// <param name="item">
500 /// <para>The item.</para>
501 /// <para></para>
502 /// </param>
503 /// <returns>
504 /// <para>The int</para>
505 /// <para></para>
506 /// </returns>
507 [MethodImpl(MethodImplOptions.AggressiveInlining)]
508 public int IndexOf(TLink item)
509 {
510     if (_equalityComparer.Equals(Index, item))
511     {
512         return _constants.IndexPart;
513     }
514     if (_equalityComparer.Equals(Source, item))
515     {
516         return _constants.SourcePart;
517     }
518     if (_equalityComparer.Equals(Target, item))
519     {
520         return _constants.TargetPart;
521     }
522     return -1;
523 }
524
525 /// <summary>
526 /// <para>
527 /// Inserts the index.
528 /// </para>
529 /// <para></para>
530 /// </summary>
531 /// <param name="index">
532 /// <para>The index.</para>
533 /// <para></para>
534 /// </param>
535 /// <param name="item">
536 /// <para>The item.</para>
537 /// <para></para>
538 /// </param>
539 [MethodImpl(MethodImplOptions.AggressiveInlining)]
540 public void Insert(int index, TLink item) => throw new NotSupportedException();
541

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory
9  {
10     /// <summary>
11     /// <para>
12     /// The links header.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct LinksHeader<TLink> : IEquatable<LinksHeader<TLink>>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
19             ↪ EqualityComparer<TLink>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<LinksHeader<TLink>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The allocated links.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLink AllocatedLinks;
36
37         /// <summary>
38         /// <para>
39         /// The reserved links.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink ReservedLinks;
44
45         /// <summary>
46         /// <para>
47         /// The free links.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public TLink FreeLinks;
52
53         /// <summary>
54         /// <para>
55         /// The first free link.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         public TLink FirstFreeLink;
60     }
61 }

```

```

56     /// <summary>
57     /// <para>
58     /// The root as source.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     public TLink RootAsSource;
63     /// <summary>
64     /// <para>
65     /// The root as target.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public TLink RootAsTarget;
70     /// <summary>
71     /// <para>
72     /// The last free link.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public TLink LastFreeLink;
77     /// <summary>
78     /// <para>
79     /// The reserved.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLink Reserved8;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equals.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="obj">
92     /// <para>The obj.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override bool Equals(object obj) => obj is LinksHeader<TLink> linksHeader ?
    ↳ Equals(linksHeader) : false;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance equals.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="other">
109    /// <para>The other.</para>
110    /// <para></para>
111    /// </param>
112    /// <returns>
113    /// <para>The bool</para>
114    /// <para></para>
115    /// </returns>
116    [MethodImpl(MethodImplOptions.AggressiveInlining)]
117    public bool Equals(LinksHeader<TLink> other)
118        => _equalityComparer.Equals(AllocatedLinks, other.AllocatedLinks)
119        && _equalityComparer.Equals(ReservedLinks, other.ReservedLinks)
120        && _equalityComparer.Equals(FreeLinks, other.FreeLinks)
121        && _equalityComparer.Equals(FirstFreeLink, other.FirstFreeLink)
122        && _equalityComparer.Equals(RootAsSource, other.RootAsSource)
123        && _equalityComparer.Equals(RootAsTarget, other.RootAsTarget)
124        && _equalityComparer.Equals(LastFreeLink, other.LastFreeLink)
125        && _equalityComparer.Equals(Reserved8, other.Reserved8);
126
127    /// <summary>
128    /// <para>
129    /// Gets the hash code.
130    /// </para>
131    /// <para></para>
132    /// </summary>

```



```

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

```

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

```

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

```

```

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

```

```

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

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

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

194
195    /// <summary>
196    /// <para>
197    /// Gets the link data part reference using the specified link.
198    /// </para>
199    /// <para></para>
200    /// </summary>
201    /// <param name="link">
202    /// <para>The link.</para>
203    /// <para></para>
204    /// </param>
205    /// <returns>
206    /// <para>A ref raw link data part of t link</para>
207    /// <para></para>
208    /// </returns>
209    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

210 protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↳ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));
211
212 /// <summary>
213 /// <para>
214 /// Gets the link index part reference using the specified link.
215 /// </para>
216 /// <para></para>
217 /// </summary>
218 /// <param name="link">
219 /// <para>The link.</para>
220 /// <para></para>
221 /// </param>
222 /// <returns>
223 /// <para>A ref raw link index part of t link</para>
224 /// <para></para>
225 /// </returns>
226 [MethodImpl(MethodImplOptions.AggressiveInlining)]
227 protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↳ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
    ↳ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
228
229 /// <summary>
230 /// <para>
231 /// Gets the link values using the specified link index.
232 /// </para>
233 /// <para></para>
234 /// </summary>
235 /// <param name="linkIndex">
236 /// <para>The link index.</para>
237 /// <para></para>
238 /// </param>
239 /// <returns>
240 /// <para>A list of t link</para>
241 /// <para></para>
242 /// </returns>
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
245 {
246     ref var link = ref GetLinkDataPartReference(linkIndex);
247     return new Link<TLink>(linkIndex, link.Source, link.Target);
248 }
249
250 /// <summary>
251 /// <para>
252 /// Determines whether this instance first is to the left of second.
253 /// </para>
254 /// <para></para>
255 /// </summary>
256 /// <param name="first">
257 /// <para>The first.</para>
258 /// <para></para>
259 /// </param>
260 /// <param name="second">
261 /// <para>The second.</para>
262 /// <para></para>
263 /// </param>
264 /// <returns>
265 /// <para>The bool</para>
266 /// <para></para>
267 /// </returns>
268 [MethodImpl(MethodImplOptions.AggressiveInlining)]
269 protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
270 {
271     ref var firstLink = ref GetLinkDataPartReference(first);
272     ref var secondLink = ref GetLinkDataPartReference(second);
273     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↳ secondLink.Source, secondLink.Target);
274 }
275
276 /// <summary>
277 /// <para>
278 /// Determines whether this instance first is to the right of second.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 /// <param name="first">

```

```

283 /// <para>The first.</para>
284 /// <para></para>
285 /// </param>
286 /// <param name="second">
287 /// <para>The second.</para>
288 /// <para></para>
289 /// </param>
290 /// <returns>
291 /// <para>The bool</para>
292 /// <para></para>
293 /// </returns>
294 [MethodImpl(MethodImplOptions.AggressiveInlining)]
295 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
296 {
297     ref var firstLink = ref GetLinkDataPartReference(first);
298     ref var secondLink = ref GetLinkDataPartReference(second);
299     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
300         ↪ secondLink.Source, secondLink.Target);
301 }
302
303 /// <summary>
304 /// <para>
305 /// The zero.
306 /// </para>
307 /// <para></para>
308 /// </summary>
309 public TLink this[TLink index]
310 {
311     [MethodImpl(MethodImplOptions.AggressiveInlining)]
312     get
313     {
314         var root = GetTreeRoot();
315         if (GreaterOrEqualThan(index, GetSize(root)))
316         {
317             return Zero;
318         }
319         while (!EqualToZero(root))
320         {
321             var left = GetLeftOrDefault(root);
322             var leftSize = GetSizeOrZero(left);
323             if (LessThan(index, leftSize))
324             {
325                 root = left;
326                 continue;
327             }
328             if (AreEqual(index, leftSize))
329             {
330                 return root;
331             }
332             root = GetRightOrDefault(root);
333             index = Subtract(index, Increment(leftSize));
334         }
335         return Zero; // TODO: Impossible situation exception (only if tree structure
336         ↪ broken)
337     }
338 }
339
340 /// <summary>
341 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
342 ↪ (концом).
343 /// </summary>
344 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
345 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
346 /// <returns>Индекс искомой связи.</returns>
347 [MethodImpl(MethodImplOptions.AggressiveInlining)]
348 public TLink Search(TLink source, TLink target)
349 {
350     var root = GetTreeRoot();
351     while (!EqualToZero(root))
352     {
353         ref var rootLink = ref GetLinkDataPartReference(root);
354         var rootSource = rootLink.Source;
355         var rootTarget = rootLink.Target;
356         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
357         ↪ node.Key < root.Key
358         {
359             root = GetLeftOrDefault(root);
360         }
361     }

```

```

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

```

```

434 /// </param>
435 /// <returns>
436 /// <para>The link</para>
437 /// <para></para>
438 /// </returns>
439 [MethodImpl(MethodImplOptions.AggressiveInlining)]
440 public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
    ↳ GetTreeRoot(), handler);
441
442 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
    ↳ low-level MSIL stack.
443 [MethodImpl(MethodImplOptions.AggressiveInlining)]
444 private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
445 {
446     var @continue = Continue;
447     if (EqualToZero(link))
448     {
449         return @continue;
450     }
451     var linkBasePart = GetBasePartValue(link);
452     var @break = Break;
453     if (GreaterThan(linkBasePart, @base))
454     {
455         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
456         {
457             return @break;
458         }
459     }
460     else if (LessThan(linkBasePart, @base))
461     {
462         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
463         {
464             return @break;
465         }
466     }
467     else //if (linkBasePart == @base)
468     {
469         if (AreEqual(handler(GetLinkValues(link)), @break))
470         {
471             return @break;
472         }
473         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
474         {
475             return @break;
476         }
477         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
478         {
479             return @break;
480         }
481     }
482     return @continue;
483 }
484
485 /// <summary>
486 /// <para>
487 /// Prints the node value using the specified node.
488 /// </para>
489 /// <para></para>
490 /// </summary>
491 /// <param name="node">
492 /// <para>The node.</para>
493 /// <para></para>
494 /// </param>
495 /// <param name="sb">
496 /// <para>The sb.</para>
497 /// <para></para>
498 /// </param>
499 [MethodImpl(MethodImplOptions.AggressiveInlining)]
500 protected override void PrintNodeValue(TLink node, StringBuilder sb)
501 {
502     ref var link = ref GetLinkDataPartReference(node);
503     sb.Append(' ');
504     sb.Append(link.Source);
505     sb.Append('-');
506     sb.Append('>');
507     sb.Append(link.Target);
508 }
509 }

```

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

```

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

```



```

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

```

```

151     protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
    ↪     rootSource, TLink rootTarget);
152
153     /// <summary>
154     /// <para>
155     /// Determines whether this instance first is to the left of second.
156     /// </para>
157     /// <para></para>
158     /// </summary>
159     /// <param name="source">
160     /// <para>The source.</para>
161     /// <para></para>
162     /// </param>
163     /// <param name="target">
164     /// <para>The target.</para>
165     /// <para></para>
166     /// </param>
167     /// <param name="rootSource">
168     /// <para>The root source.</para>
169     /// <para></para>
170     /// </param>
171     /// <param name="rootTarget">
172     /// <para>The root target.</para>
173     /// <para></para>
174     /// </param>
175     /// <returns>
176     /// <para>The bool</para>
177     /// <para></para>
178     /// </returns>
179     [MethodImpl(MethodImplOptions.AggressiveInlining)]
180     protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
    ↪     rootSource, TLink rootTarget);
181
182     /// <summary>
183     /// <para>
184     /// Gets the header reference.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <returns>
189     /// <para>A ref links header of t link</para>
190     /// <para></para>
191     /// </returns>
192     [MethodImpl(MethodImplOptions.AggressiveInlining)]
193     protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪     AsRef<LinksHeader<TLink>>(Header);
194
195     /// <summary>
196     /// <para>
197     /// Gets the link data part reference using the specified link.
198     /// </para>
199     /// <para></para>
200     /// </summary>
201     /// <param name="link">
202     /// <para>The link.</para>
203     /// <para></para>
204     /// </param>
205     /// <returns>
206     /// <para>A ref raw link data part of t link</para>
207     /// <para></para>
208     /// </returns>
209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
210     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↪     AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↪     _addressToInt64Converter.Convert(link)));
211
212     /// <summary>
213     /// <para>
214     /// Gets the link index part reference using the specified link.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="link">
219     /// <para>The link.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>

```

```

223 /// <para>A ref raw link index part of t link</para>
224 /// <para></para>
225 /// </returns>
226 [MethodImpl(MethodImplOptions.AggressiveInlining)]
227 protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
228     ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
229     ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
230
231 /// <summary>
232 /// <para>
233 /// Gets the link values using the specified link index.
234 /// </para>
235 /// <para></para>
236 /// </summary>
237 /// <param name="linkIndex">
238 /// <para>The link index.</para>
239 /// <para></para>
240 /// </param>
241 /// <returns>
242 /// <para>A list of t link</para>
243 /// <para></para>
244 /// </returns>
245 [MethodImpl(MethodImplOptions.AggressiveInlining)]
246 protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
247 {
248     ref var link = ref GetLinkDataPartReference(linkIndex);
249     return new Link<TLink>(linkIndex, link.Source, link.Target);
250 }
251
252 /// <summary>
253 /// <para>
254 /// Determines whether this instance first is to the left of second.
255 /// </para>
256 /// <para></para>
257 /// </summary>
258 /// <param name="first">
259 /// <para>The first.</para>
260 /// <para></para>
261 /// </param>
262 /// <param name="second">
263 /// <para>The second.</para>
264 /// <para></para>
265 /// </param>
266 /// <returns>
267 /// <para>The bool</para>
268 /// <para></para>
269 /// </returns>
270 [MethodImpl(MethodImplOptions.AggressiveInlining)]
271 protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
272 {
273     ref var firstLink = ref GetLinkDataPartReference(first);
274     ref var secondLink = ref GetLinkDataPartReference(second);
275     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
276     ↪ secondLink.Source, secondLink.Target);
277 }
278
279 /// <summary>
280 /// <para>
281 /// Determines whether this instance first is to the right of second.
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 FirstIsToTheRightOfSecond(TLink first, TLink second)
299 {
300     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 TLink this[TLink index]
309 {
310     [MethodImpl(MethodImplOptions.AggressiveInlining)]
311     get
312     {
313         var root = GetTreeRoot();
314         if (GreaterOrEqualThan(index, GetSize(root)))
315         {
316             return Zero;
317         }
318         while (!EqualToZero(root))
319         {
320             var left = GetLeftOrDefault(root);
321             var leftSize = GetSizeOrZero(left);
322             if (LessThan(index, leftSize))
323             {
324                 root = left;
325                 continue;
326             }
327             if (AreEqual(index, leftSize))
328             {
329                 return root;
330             }
331             root = GetRightOrDefault(root);
332             index = Subtract(index, Increment(leftSize));
333         }
334         return Zero; // TODO: Impossible situation exception (only if tree structure
    ↪ 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 TLink Search(TLink source, TLink target)
346 {
347     var root = GetTreeRoot();
348     while (!EqualToZero(root))
349     {
350         ref var rootLink = ref GetLinkDataPartReference(root);
351         var rootSource = rootLink.Source;
352         var rootTarget = rootLink.Target;
353         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
    ↪ node.Key < root.Key
354         {
355             root = GetLeftOrDefault(root);
356         }
357         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
    ↪ node.Key > root.Key
358         {
359             root = GetRightOrDefault(root);
360         }
361         else // node.Key == root.Key
362         {
363             return root;
364         }
365     }
366     return Zero;
367 }
368
369 // TODO: Return indices range instead of references count
370 /// <summary>

```

```

371    /// <para>
372    /// Counts the usages using the specified link.
373    /// </para>
374    /// <para></para>
375    /// </summary>
376    /// <param name="link">
377    /// <para>The link.</para>
378    /// <para></para>
379    /// </param>
380    /// <returns>
381    /// <para>The link</para>
382    /// <para></para>
383    /// </returns>
384    [MethodImpl(MethodImplOptions.AggressiveInlining)]
385    public TLink CountUsages(TLink link)
386    {
387        var root = GetTreeRoot();
388        var total = GetSize(root);
389        var totalRightIgnore = Zero;
390        while (!EqualToZero(root))
391        {
392            var @base = GetBasePartValue(root);
393            if (LessOrEqualThan(@base, link))
394            {
395                root = GetRightOrDefault(root);
396            }
397            else
398            {
399                totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
400                root = GetLeftOrDefault(root);
401            }
402        }
403        root = GetTreeRoot();
404        var totalLeftIgnore = Zero;
405        while (!EqualToZero(root))
406        {
407            var @base = GetBasePartValue(root);
408            if (GreaterOrEqualThan(@base, link))
409            {
410                root = GetLeftOrDefault(root);
411            }
412            else
413            {
414                totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
415                root = GetRightOrDefault(root);
416            }
417        }
418        return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
419    }
420
421    /// <summary>
422    /// <para>
423    /// Eaches the usage using the specified base.
424    /// </para>
425    /// <para></para>
426    /// </summary>
427    /// <param name="@base">
428    /// <para>The base.</para>
429    /// <para></para>
430    /// </param>
431    /// <param name="handler">
432    /// <para>The handler.</para>
433    /// <para></para>
434    /// </param>
435    /// <returns>
436    /// <para>The link</para>
437    /// <para></para>
438    /// </returns>
439    [MethodImpl(MethodImplOptions.AggressiveInlining)]
440    public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
441        ↪ 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 TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
447    {
448        var @continue = Continue;

```

```

447         if (EqualToZero(link))
448         {
449             return @continue;
450         }
451         var linkBasePart = GetBasePartValue(link);
452         var @break = Break;
453         if (GreaterThan(linkBasePart, @base))
454         {
455             if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
456             {
457                 return @break;
458             }
459         }
460         else if (LessThan(linkBasePart, @base))
461         {
462             if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
463             {
464                 return @break;
465             }
466         }
467         else //if (linkBasePart == @base)
468         {
469             if (AreEqual(handler(GetLinkValues(link)), @break))
470             {
471                 return @break;
472             }
473             if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
474             {
475                 return @break;
476             }
477             if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
478             {
479                 return @break;
480             }
481         }
482         return @continue;
483     }

```

```

485     /// <summary>
486     /// <para>
487     /// Prints the node value using the specified node.
488     /// </para>
489     /// <para></para>
490     /// </summary>
491     /// <param name="node">
492     /// <para>The node.</para>
493     /// <para></para>
494     /// </param>
495     /// <param name="sb">
496     /// <para>The sb.</para>
497     /// <para></para>
498     /// </param>

```

```

499     [MethodImpl(MethodImplOptions.AggressiveInlining)]
500     protected override void PrintNodeValue(TLink node, StringBuilder sb)
501     {
502         ref var link = ref GetLinkDataPartReference(node);
503         sb.Append(' ');
504         sb.Append(link.Source);
505         sb.Append('-');
506         sb.Append('>');
507         sb.Append(link.Target);
508     }
509 }
510 }

```

## 1.34 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTree

```

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

```

```

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

```

```

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

```



```

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

```

```

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

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

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

```

### 1.35 ./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{TLink}"/>
14    public unsafe class ExternalLinksSourcesSizeBalancedTreeMethods<TLink> :
        ↳ ExternalLinksSizeBalancedTreeMethodsBase<TLink>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="ExternalLinksSourcesSizeBalancedTreeMethods"/> instance.
19        /// </para>

```

```

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

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

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

74
75    /// <summary>
76    /// <para>
77    /// Gets the left using the specified node.
78    /// </para>
79    /// <para></para>
80    /// </summary>
81    /// <param name="node">
82    /// <para>The node.</para>
83    /// <para></para>
84    /// </param>
85    /// <returns>
86    /// <para>The link</para>
87    /// <para></para>
88    /// </returns>
89    [MethodImpl(MethodImplOptions.AggressiveInlining)]
90    protected override TLink GetLeft(TLink node) =>
    ↪ GetLinkIndexPartReference(node).LeftAsSource;
91

```

```

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

```

```

166     /// <param name="node">
167     /// <para>The node.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="size">
171     /// <para>The size.</para>
172     /// <para></para>
173     /// </param>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     protected override void SetSize(TLink node, TLink size) =>
176     ↪ GetLinkIndexPartReference(node).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 TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
190
191     /// <summary>
192     /// <para>
193     /// Gets the base part value using the specified link.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="link">
198     /// <para>The link.</para>
199     /// <para></para>
200     /// </param>
201     /// <returns>
202     /// <para>The link</para>
203     /// <para></para>
204     /// </returns>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override TLink GetBasePartValue(TLink link) =>
207     ↪ GetLinkDataPartReference(link).Source;
208
209     /// <summary>
210     /// <para>
211     /// Determines whether this instance first is to the left of second.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="firstSource">
216     /// <para>The first source.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="firstTarget">
220     /// <para>The first target.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="secondSource">
224     /// <para>The second source.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="secondTarget">
228     /// <para>The second target.</para>
229     /// <para></para>
230     /// </param>
231     /// <returns>
232     /// <para>The bool</para>
233     /// <para></para>
234     /// </returns>
235     [MethodImpl(MethodImplOptions.AggressiveInlining)]
236     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
237     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
238     ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>

```

```

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

```

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

```

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

```

```

28     /// <para></para>
29     /// </param>
30     /// <param name="linksIndexParts">
31     /// <para>A links index parts.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="header">
35     /// <para>A header.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
        ↪ base(constants, linksDataParts, linksIndexParts, header) { }

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

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

74
75     /// <summary>
76     /// <para>
77     /// Gets the left using the specified node.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <param name="node">
82     /// <para>The node.</para>
83     /// <para></para>
84     /// </param>
85     /// <returns>
86     /// <para>The link</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     protected override TLink GetLeft(TLink node) =>
        ↪ GetLinkIndexPartReference(node).LeftAsTarget;

91
92     /// <summary>
93     /// <para>
94     /// Gets the right using the specified node.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <param name="node">
99     /// <para>The node.</para>

```

```

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

```



```

174 [MethodImpl(MethodImplOptions.AggressiveInlining)]
175 protected override void SetSize(TLink node, TLink size) =>
    ↳ GetLinkIndexPartReference(node).SizeAsTarget = size;

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

206
207 /// <summary>
208 /// <para>
209 /// Determines whether this instance first is to the left of second.
210 /// </para>
211 /// <para></para>
212 /// </summary>
213 /// <param name="firstSource">
214 /// <para>The first source.</para>
215 /// <para></para>
216 /// </param>
217 /// <param name="firstTarget">
218 /// <para>The first target.</para>
219 /// <para></para>
220 /// </param>
221 /// <param name="secondSource">
222 /// <para>The second source.</para>
223 /// <para></para>
224 /// </param>
225 /// <param name="secondTarget">
226 /// <para>The second target.</para>
227 /// <para></para>
228 /// </param>
229 /// <returns>
230 /// <para>The bool</para>
231 /// <para></para>
232 /// </returns>
233 [MethodImpl(MethodImplOptions.AggressiveInlining)]
234 protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↳ TLink secondSource, TLink 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(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
264
265     /// <summary>
266     /// <para>
267     /// Clears the node using the specified node.
268     /// </para>
269     /// <para></para>
270     /// </summary>
271     /// <param name="node">
272     /// <para>The node.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLink node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);
279         link.LeftAsTarget = Zero;
280         link.RightAsTarget = Zero;
281         link.SizeAsTarget = Zero;
282     }
283 }
284 }

```

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

```

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

```

```

37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
    ↪     byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
    ↪     linksDataParts, linksIndexParts, header) { }

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

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

74
75     /// <summary>
76     /// <para>
77     /// Gets the left using the specified node.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <param name="node">
82     /// <para>The node.</para>
83     /// <para></para>
84     /// </param>
85     /// <returns>
86     /// <para>The link</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     protected override TLink GetLeft(TLink node) =>
    ↪     GetLinkIndexPartReference(node).LeftAsTarget;

91
92     /// <summary>
93     /// <para>
94     /// Gets the right using the specified node.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <param name="node">
99     /// <para>The node.</para>
100    /// <para></para>
101    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLink GetRight(TLink node) =>
    ↪     GetLinkIndexPartReference(node).RightAsTarget;

```

```

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

```

```

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

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

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

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

```

```

257     /// </param>
258     /// <returns>
259     /// <para>The bool</para>
260     /// <para></para>
261     /// </returns>
262     [MethodImpl(MethodImplOptions.AggressiveInlining)]
263     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
264
265     /// <summary>
266     /// <para>
267     /// Clears the node using the specified node.
268     /// </para>
269     /// <para></para>
270     /// </summary>
271     /// <param name="node">
272     /// <para>The node.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLink node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);
279         link.LeftAsTarget = Zero;
280         link.RightAsTarget = Zero;
281         link.SizeAsTarget = Zero;
282     }
283 }
284 }

```

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

```

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

```

```

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

```

```

122    /// <para>The link</para>
123    /// <para></para>
124    /// </returns>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected abstract TLink GetBasePartValue(TLink link);
127
128    /// <summary>
129    /// <para>
130    /// Gets the key part value using the specified link.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    /// <param name="link">
135    /// <para>The link.</para>
136    /// <para></para>
137    /// </param>
138    /// <returns>
139    /// <para>The link</para>
140    /// <para></para>
141    /// </returns>
142    [MethodImpl(MethodImplOptions.AggressiveInlining)]
143    protected abstract TLink GetKeyPartValue(TLink link);
144
145    /// <summary>
146    /// <para>
147    /// Gets the link data part reference using the specified link.
148    /// </para>
149    /// <para></para>
150    /// </summary>
151    /// <param name="link">
152    /// <para>The link.</para>
153    /// <para></para>
154    /// </param>
155    /// <returns>
156    /// <para>A ref raw link data part of t link</para>
157    /// <para></para>
158    /// </returns>
159    [MethodImpl(MethodImplOptions.AggressiveInlining)]
160    protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↪ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));
161
162    /// <summary>
163    /// <para>
164    /// Gets the link index part reference using the specified link.
165    /// </para>
166    /// <para></para>
167    /// </summary>
168    /// <param name="link">
169    /// <para>The link.</para>
170    /// <para></para>
171    /// </param>
172    /// <returns>
173    /// <para>A ref raw link index part of t link</para>
174    /// <para></para>
175    /// </returns>
176    [MethodImpl(MethodImplOptions.AggressiveInlining)]
177    protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
    ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
178
179    /// <summary>
180    /// <para>
181    /// Determines whether this instance first is to the left of second.
182    /// </para>
183    /// <para></para>
184    /// </summary>
185    /// <param name="first">
186    /// <para>The first.</para>
187    /// <para></para>
188    /// </param>
189    /// <param name="second">
190    /// <para>The second.</para>
191    /// <para></para>
192    /// </param>
193    /// <returns>
194    /// <para>The bool</para>
195    /// <para></para>

```



```

196     /// </returns>
197     [MethodImpl(MethodImplOptions.AggressiveInlining)]
198     protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second) =>
199         ↳ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
200
201     /// <summary>
202     /// <para>
203     /// Determines whether this instance first is to the right of second.
204     /// </para>
205     /// </summary>
206     /// <param name="first">
207     /// <para>The first.</para>
208     /// </param>
209     /// <param name="second">
210     /// <para>The second.</para>
211     /// </param>
212     /// </returns>
213     /// <para>The bool</para>
214     /// </returns>
215     [MethodImpl(MethodImplOptions.AggressiveInlining)]
216     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
217         ↳ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
218
219     /// <summary>
220     /// <para>
221     /// Gets the link values using the specified link index.
222     /// </para>
223     /// </summary>
224     /// <param name="linkIndex">
225     /// <para>The link index.</para>
226     /// </param>
227     /// </returns>
228     /// <para>A list of t link</para>
229     /// </returns>
230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
231     protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
232     {
233         ref var link = ref GetLinkDataPartReference(linkIndex);
234         return new Link<TLink>(linkIndex, link.Source, link.Target);
235     }
236
237     /// <summary>
238     /// <para>
239     /// The zero.
240     /// </para>
241     /// </summary>
242     public TLink this[TLink link, TLink index]
243     {
244         [MethodImpl(MethodImplOptions.AggressiveInlining)]
245         get
246         {
247             var root = GetTreeRoot(link);
248             if (GreaterOrEqualThan(index, GetSize(root)))
249             {
250                 return Zero;
251             }
252             while (!EqualToZero(root))
253             {
254                 var left = GetLeftOrDefault(root);
255                 var leftSize = GetSizeOrZero(left);
256                 if (LessThan(index, leftSize))
257                 {
258                     root = left;
259                     continue;
260                 }
261                 if (AreEqual(index, leftSize))
262                 {
263                     return root;
264                 }
265                 root = GetRightOrDefault(root);
266             }
267         }
268     }

```

```

272         index = Subtract(index, Increment(leftSize));
273     }
274     return Zero; // TODO: Impossible situation exception (only if tree structure
    ↪ broken)
275 }
276 }
277
278 /// <summary>
279 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
280 /// </summary>
281 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
282 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
283 /// <returns>Индекс искомой связи.</returns>
284 [MethodImpl(MethodImplOptions.AggressiveInlining)]
285 public abstract TLink Search(TLink source, TLink target);
286
287 /// <summary>
288 /// <para>
289 /// Searches the core using the specified root.
290 /// </para>
291 /// <para></para>
292 /// </summary>
293 /// <param name="root">
294 /// <para>The root.</para>
295 /// <para></para>
296 /// </param>
297 /// <param name="key">
298 /// <para>The key.</para>
299 /// <para></para>
300 /// </param>
301 /// <returns>
302 /// <para>The zero.</para>
303 /// <para></para>
304 /// </returns>
305 [MethodImpl(MethodImplOptions.AggressiveInlining)]
306 protected TLink SearchCore(TLink root, TLink key)
307 {
308     while (!EqualToZero(root))
309     {
310         var rootKey = GetKeyPartValue(root);
311         if (LessThan(key, rootKey)) // node.Key < root.Key
312         {
313             root = GetLeftOrDefault(root);
314         }
315         else if (GreaterThan(key, rootKey)) // node.Key > root.Key
316         {
317             root = GetRightOrDefault(root);
318         }
319         else // node.Key == root.Key
320         {
321             return root;
322         }
323     }
324     return Zero;
325 }
326
327 // TODO: Return indices range instead of references count
328 /// <summary>
329 /// <para>
330 /// Counts the usages using the specified link.
331 /// </para>
332 /// <para></para>
333 /// </summary>
334 /// <param name="link">
335 /// <para>The link.</para>
336 /// <para></para>
337 /// </param>
338 /// <returns>
339 /// <para>The link</para>
340 /// <para></para>
341 /// </returns>
342 [MethodImpl(MethodImplOptions.AggressiveInlining)]
343 public TLink CountUsages(TLink link) => GetSizeOrZero(GetTreeRoot(link));
344
345 /// <summary>
346 /// <para>
347 /// Eaches the usage using the specified base.

```

```

348     /// </para>
349     /// <para></para>
350     /// </summary>
351     /// <param name="@base">
352     /// <para>The base.</para>
353     /// <para></para>
354     /// </param>
355     /// <param name="handler">
356     /// <para>The handler.</para>
357     /// <para></para>
358     /// </param>
359     /// <returns>
360     /// <para>The link</para>
361     /// <para></para>
362     /// </returns>
363     [MethodImpl(MethodImplOptions.AggressiveInlining)]
364     public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
        ↪ GetTreeRoot(@base), handler);
365
366     // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
        ↪ low-level MSIL stack.
367     [MethodImpl(MethodImplOptions.AggressiveInlining)]
368     private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
369     {
370         var @continue = Continue;
371         if (EqualToZero(link))
372         {
373             return @continue;
374         }
375         var @break = Break;
376         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
377         {
378             return @break;
379         }
380         if (AreEqual(handler(GetLinkValues(link)), @break))
381         {
382             return @break;
383         }
384         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
385         {
386             return @break;
387         }
388         return @continue;
389     }
390
391     /// <summary>
392     /// <para>
393     /// Prints the node value using the specified node.
394     /// </para>
395     /// <para></para>
396     /// </summary>
397     /// <param name="node">
398     /// <para>The node.</para>
399     /// <para></para>
400     /// </param>
401     /// <param name="sb">
402     /// <para>The sb.</para>
403     /// <para></para>
404     /// </param>
405     [MethodImpl(MethodImplOptions.AggressiveInlining)]
406     protected override void PrintNodeValue(TLink node, StringBuilder sb)
407     {
408         ref var link = ref GetLinkDataPartReference(node);
409         sb.Append(' ');
410         sb.Append(link.Source);
411         sb.Append(' - ');
412         sb.Append(' > ');
413         sb.Append(link.Target);
414     }
415 }
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{TLink}"/>
21     /// <seealso cref="ILinksTreeMethods{TLink}"/>
22     public unsafe abstract class InternalLinksSizeBalancedTreeMethodsBase<TLink> :
23         ↳ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
26             ↳ UncheckedConverter<TLink, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLink Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLink Continue;
43
44         /// <summary>
45         /// <para>
46         /// The links data parts.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly byte* LinksDataParts;
51
52         /// <summary>
53         /// <para>
54         /// The links index parts.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         protected readonly byte* LinksIndexParts;
59
60         /// <summary>
61         /// <para>
62         /// The header.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         protected readonly byte* Header;
67
68         /// <summary>
69         /// <para>
70         /// Initializes a new <see cref="InternalLinksSizeBalancedTreeMethodsBase"/> instance.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <param name="constants">
75         /// <para>A constants.</para>
76         /// <para></para>
77         /// </param>
78         /// <param name="linksDataParts">
79         /// <para>A links data parts.</para>
80         /// <para></para>
81         /// </param>
82         /// <param name="linksIndexParts">
83         /// <para>A links index parts.</para>
84         /// <para></para>
85         /// </param>
86         /// <param name="header">
87         /// <para>A header.</para>
88         /// <para></para>
89         /// </param>

```

```

84 [MethodImpl(MethodImplOptions.AggressiveInlining)]
85 protected InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
    ↳ byte* linksDataParts, byte* linksIndexParts, byte* header)
86 {
87     LinksDataParts = linksDataParts;
88     LinksIndexParts = linksIndexParts;
89     Header = header;
90     Break = constants.Break;
91     Continue = constants.Continue;
92 }
93
94 /// <summary>
95 /// <para>
96 /// Gets the tree root using the specified link.
97 /// </para>
98 /// <para></para>
99 /// </summary>
100 /// <param name="link">
101 /// <para>The link.</para>
102 /// <para></para>
103 /// </param>
104 /// <returns>
105 /// <para>The link</para>
106 /// <para></para>
107 /// </returns>
108 [MethodImpl(MethodImplOptions.AggressiveInlining)]
109 protected abstract TLink GetTreeRoot(TLink link);
110
111 /// <summary>
112 /// <para>
113 /// Gets the base part value using the specified link.
114 /// </para>
115 /// <para></para>
116 /// </summary>
117 /// <param name="link">
118 /// <para>The link.</para>
119 /// <para></para>
120 /// </param>
121 /// <returns>
122 /// <para>The link</para>
123 /// <para></para>
124 /// </returns>
125 [MethodImpl(MethodImplOptions.AggressiveInlining)]
126 protected abstract TLink GetBasePartValue(TLink link);
127
128 /// <summary>
129 /// <para>
130 /// Gets the key part value using the specified link.
131 /// </para>
132 /// <para></para>
133 /// </summary>
134 /// <param name="link">
135 /// <para>The link.</para>
136 /// <para></para>
137 /// </param>
138 /// <returns>
139 /// <para>The link</para>
140 /// <para></para>
141 /// </returns>
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 protected abstract TLink GetKeyPartValue(TLink link);
144
145 /// <summary>
146 /// <para>
147 /// Gets the link data part reference using the specified link.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <param name="link">
152 /// <para>The link.</para>
153 /// <para></para>
154 /// </param>
155 /// <returns>
156 /// <para>A ref raw link data part of t link</para>
157 /// <para></para>
158 /// </returns>
159 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

160 protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
161     ↳ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
162     ↳ _addressToInt64Converter.Convert(link)));
163
164 /// <summary>
165 /// <para>
166 /// Gets the link index part reference using the specified link.
167 /// </para>
168 /// <para></para>
169 /// </summary>
170 /// <param name="link">
171 /// <para>The link.</para>
172 /// <para></para>
173 /// </param>
174 /// <returns>
175 /// <para>A ref raw link index part of t link</para>
176 /// <para></para>
177 /// </returns>
178 [MethodImpl(MethodImplOptions.AggressiveInlining)]
179 protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
180     ↳ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
181     ↳ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
182
183 /// <summary>
184 /// <para>
185 /// Determines whether this instance first is to the left of second.
186 /// </para>
187 /// <para></para>
188 /// </summary>
189 /// <param name="first">
190 /// <para>The first.</para>
191 /// <para></para>
192 /// </param>
193 /// <param name="second">
194 /// <para>The second.</para>
195 /// <para></para>
196 /// </param>
197 /// <returns>
198 /// <para>The bool</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
203     ↳ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
204
205 /// <summary>
206 /// <para>
207 /// Determines whether this instance first is to the right of second.
208 /// </para>
209 /// <para></para>
210 /// </summary>
211 /// <param name="first">
212 /// <para>The first.</para>
213 /// <para></para>
214 /// </param>
215 /// <param name="second">
216 /// <para>The second.</para>
217 /// <para></para>
218 /// </param>
219 /// <returns>
220 /// <para>The bool</para>
221 /// <para></para>
222 /// </returns>
223 [MethodImpl(MethodImplOptions.AggressiveInlining)]
224 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
225     ↳ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
226
227 /// <summary>
228 /// <para>
229 /// Gets the link values using the specified link index.
230 /// </para>
231 /// <para></para>
232 /// </summary>
233 /// <param name="linkIndex">
234 /// <para>The link index.</para>
235 /// <para></para>
236 /// </param>
237 /// <returns>

```

```

232 /// <para>A list of t link</para>
233 /// <para></para>
234 /// </returns>
235 [MethodImpl(MethodImplOptions.AggressiveInlining)]
236 protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
237 {
238     ref var link = ref GetLinkDataPartReference(linkIndex);
239     return new Link<TLink>(linkIndex, link.Source, link.Target);
240 }
241
242 /// <summary>
243 /// <para>
244 /// The zero.
245 /// </para>
246 /// <para></para>
247 /// </summary>
248 public TLink this[TLink link, TLink index]
249 {
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     get
252     {
253         var root = GetTreeRoot(link);
254         if (GreaterOrEqualThan(index, GetSize(root)))
255         {
256             return Zero;
257         }
258         while (!EqualToZero(root))
259         {
260             var left = GetLeftOrDefault(root);
261             var leftSize = GetSizeOrZero(left);
262             if (LessThan(index, leftSize))
263             {
264                 root = left;
265                 continue;
266             }
267             if (AreEqual(index, leftSize))
268             {
269                 return root;
270             }
271             root = GetRightOrDefault(root);
272             index = Subtract(index, Increment(leftSize));
273         }
274         return Zero; // TODO: Impossible situation exception (only if tree structure
275             ↳ broken)
276     }
277 }
278
279 /// <summary>
280 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
281 ↳ (концом).
282 /// </summary>
283 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
284 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
285 /// <returns>Индекс искомой связи.</returns>
286 [MethodImpl(MethodImplOptions.AggressiveInlining)]
287 public abstract TLink Search(TLink source, TLink target);
288
289 /// <summary>
290 /// <para>
291 /// Searches the core using the specified root.
292 /// </para>
293 /// <para></para>
294 /// </summary>
295 /// <param name="root">
296 /// <para>The root.</para>
297 /// <para></para>
298 /// </param>
299 /// <param name="key">
300 /// <para>The key.</para>
301 /// <para></para>
302 /// </param>
303 /// <returns>
304 /// <para>The zero.</para>
305 /// <para></para>
306 /// </returns>
307 [MethodImpl(MethodImplOptions.AggressiveInlining)]
308 protected TLink SearchCore(TLink root, TLink key)
309 {

```

```

308 while (!EqualToZero(root))
309 {
310     var rootKey = GetKeyPartValue(root);
311     if (LessThan(key, rootKey)) // node.Key < root.Key
312     {
313         root = GetLeftOrDefault(root);
314     }
315     else if (GreaterThan(key, rootKey)) // node.Key > root.Key
316     {
317         root = GetRightOrDefault(root);
318     }
319     else // node.Key == root.Key
320     {
321         return root;
322     }
323 }
324 return Zero;
325 }
326
327 // TODO: Return indices range instead of references count
328 /// <summary>
329 /// <para>
330 /// Counts the usages using the specified link.
331 /// </para>
332 /// <para></para>
333 /// </summary>
334 /// <param name="link">
335 /// <para>The link.</para>
336 /// <para></para>
337 /// </param>
338 /// <returns>
339 /// <para>The link</para>
340 /// <para></para>
341 /// </returns>
342 [MethodImpl(MethodImplOptions.AggressiveInlining)]
343 public TLink CountUsages(TLink link) => GetSizeOrZero(GetTreeRoot(link));
344
345 /// <summary>
346 /// <para>
347 /// Eaches the usage using the specified base.
348 /// </para>
349 /// <para></para>
350 /// </summary>
351 /// <param name="@base">
352 /// <para>The base.</para>
353 /// <para></para>
354 /// </param>
355 /// <param name="handler">
356 /// <para>The handler.</para>
357 /// <para></para>
358 /// </param>
359 /// <returns>
360 /// <para>The link</para>
361 /// <para></para>
362 /// </returns>
363 [MethodImpl(MethodImplOptions.AggressiveInlining)]
364 public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
    ↳ GetTreeRoot(@base), handler);
365
366 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
    ↳ low-level MSIL stack.
367 [MethodImpl(MethodImplOptions.AggressiveInlining)]
368 private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
369 {
370     var @continue = Continue;
371     if (EqualToZero(link))
372     {
373         return @continue;
374     }
375     var @break = Break;
376     if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
377     {
378         return @break;
379     }
380     if (AreEqual(handler(GetLinkValues(link)), @break))
381     {
382         return @break;
383     }

```



```

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

```

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

```

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

```

```

43    /// </para>
44    /// <para></para>
45    /// </summary>
46    /// <param name="constants">
47    /// <para>A constants.</para>
48    /// <para></para>
49    /// </param>
50    /// <param name="linksDataParts">
51    /// <para>A links data parts.</para>
52    /// <para></para>
53    /// </param>
54    /// <param name="linksIndexParts">
55    /// <para>A links index parts.</para>
56    /// <para></para>
57    /// </param>
58    [MethodImpl(MethodImplOptions.AggressiveInlining)]
59    public InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants, byte*
    ↳ linksDataParts, byte* linksIndexParts)
60    {
61        _linksDataParts = linksDataParts;
62        _linksIndexParts = linksIndexParts;
63        Break = constants.Break;
64        Continue = constants.Continue;
65    }
66
67    /// <summary>
68    /// <para>
69    /// Gets the link data part reference using the specified link.
70    /// </para>
71    /// <para></para>
72    /// </summary>
73    /// <param name="link">
74    /// <para>The link.</para>
75    /// <para></para>
76    /// </param>
77    /// <returns>
78    /// <para>A ref raw link data part of t link</para>
79    /// <para></para>
80    /// </returns>
81    [MethodImpl(MethodImplOptions.AggressiveInlining)]
82    protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↳ AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (RawLinkDataPart<TLink>.SizeInBytes
    ↳ * _addressToInt64Converter.Convert(link)));
83
84    /// <summary>
85    /// <para>
86    /// Gets the link index part reference using the specified link.
87    /// </para>
88    /// <para></para>
89    /// </summary>
90    /// <param name="link">
91    /// <para>The link.</para>
92    /// <para></para>
93    /// </param>
94    /// <returns>
95    /// <para>A ref raw link index part of t link</para>
96    /// <para></para>
97    /// </returns>
98    [MethodImpl(MethodImplOptions.AggressiveInlining)]
99    protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↳ ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
    ↳ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
100
101    /// <summary>
102    /// <para>
103    /// Gets the first using the specified head.
104    /// </para>
105    /// <para></para>
106    /// </summary>
107    /// <param name="head">
108    /// <para>The head.</para>
109    /// <para></para>
110    /// </param>
111    /// <returns>
112    /// <para>The link</para>
113    /// <para></para>
114    /// </returns>
115    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

116     protected override TLink GetFirst(TLink head) =>
117         ↪ GetLinkIndexPartReference(head).RootAsSource;
118
119     /// <summary>
120     /// <para>
121     /// Gets the last using the specified head.
122     /// </para>
123     /// </summary>
124     /// <param name="head">
125     /// <para>The head.</para>
126     /// </param>
127     /// </returns>
128     /// <para>The link</para>
129     /// </returns>
130     [MethodImpl(MethodImplOptions.AggressiveInlining)]
131     protected override TLink GetLast(TLink head)
132     {
133         var first = GetLinkIndexPartReference(head).RootAsSource;
134         if (EqualToZero(first))
135         {
136             return first;
137         }
138         else
139         {
140             return GetPrevious(first);
141         }
142     }
143
144     /// <summary>
145     /// <para>
146     /// Gets the previous using the specified element.
147     /// </para>
148     /// </summary>
149     /// <param name="element">
150     /// <para>The element.</para>
151     /// </param>
152     /// </returns>
153     /// <para>The link</para>
154     /// </returns>
155     [MethodImpl(MethodImplOptions.AggressiveInlining)]
156     protected override TLink GetPrevious(TLink element) =>
157         ↪ GetLinkIndexPartReference(element).LeftAsSource;
158
159     /// <summary>
160     /// <para>
161     /// Gets the next using the specified element.
162     /// </para>
163     /// </summary>
164     /// <param name="element">
165     /// <para>The element.</para>
166     /// </param>
167     /// </returns>
168     /// <para>The link</para>
169     /// </returns>
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     protected override TLink GetNext(TLink element) =>
172         ↪ GetLinkIndexPartReference(element).RightAsSource;
173
174     /// <summary>
175     /// <para>
176     /// Gets the size using the specified head.
177     /// </para>
178     /// </summary>
179     /// <param name="head">
180     /// <para>The head.</para>
181     /// </param>
182     /// </returns>

```

```

191    /// <para>The link</para>
192    /// <para></para>
193    /// </returns>
194    [MethodImpl(MethodImplOptions.AggressiveInlining)]
195    protected override TLink GetSize(TLink head) =>
196        ↪ GetLinkIndexPartReference(head).SizeAsSource;
197
198    /// <summary>
199    /// <para>
200    /// Sets the first using the specified head.
201    /// </para>
202    /// <para></para>
203    /// </summary>
204    /// <param name="head">
205    /// <para>The head.</para>
206    /// <para></para>
207    /// </param>
208    /// <param name="element">
209    /// <para>The element.</para>
210    /// <para></para>
211    /// </param>
212    [MethodImpl(MethodImplOptions.AggressiveInlining)]
213    protected override void SetFirst(TLink head, TLink element) =>
214        ↪ GetLinkIndexPartReference(head).RootAsSource = element;
215
216    /// <summary>
217    /// <para>
218    /// Sets the last using the specified head.
219    /// </para>
220    /// <para></para>
221    /// </summary>
222    /// <param name="head">
223    /// <para>The head.</para>
224    /// <para></para>
225    /// </param>
226    /// <param name="element">
227    /// <para>The element.</para>
228    /// <para></para>
229    /// </param>
230    [MethodImpl(MethodImplOptions.AggressiveInlining)]
231    protected override void SetLast(TLink head, TLink element)
232    {
233        //var first = GetLinkIndexPartReference(head).RootAsSource;
234        //if (EqualToZero(first))
235        //{
236            SetFirst(head, element);
237        //}
238        //else
239        //{
240            SetPrevious(first, element);
241        //}
242    }
243
244    /// <summary>
245    /// <para>
246    /// Sets the previous using the specified element.
247    /// </para>
248    /// <para></para>
249    /// </summary>
250    /// <param name="element">
251    /// <para>The element.</para>
252    /// <para></para>
253    /// </param>
254    /// <param name="previous">
255    /// <para>The previous.</para>
256    /// <para></para>
257    /// </param>
258    [MethodImpl(MethodImplOptions.AggressiveInlining)]
259    protected override void SetPrevious(TLink element, TLink previous) =>
260        ↪ GetLinkIndexPartReference(element).LeftAsSource = previous;
261
262    /// <summary>
263    /// <para>
264    /// Sets the next using the specified element.
265    /// </para>
266    /// <para></para>
267    /// </summary>
268    /// <param name="element">

```

```

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

```

```

342     /// <para>The handler.</para>
343     /// <para></para>
344     /// </param>
345     /// <returns>
346     /// <para>The continue.</para>
347     /// <para></para>
348     /// </returns>
349     [MethodImpl(MethodImplOptions.AggressiveInlining)]
350     public TLink EachUsage(TLink source, ReadHandler<TLink>? handler)
351     {
352         var @continue = Continue;
353         var @break = Break;
354         var current = GetFirst(source);
355         var first = current;
356         while (!EqualToZero(current))
357         {
358             if (AreEqual(handler(GetLinkValues(current)), @break))
359             {
360                 return @break;
361             }
362             current = GetNext(current);
363             if (AreEqual(current, first))
364             {
365                 return @continue;
366             }
367         }
368         return @continue;
369     }
370 }
371 }

```

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

```

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

```

```

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

```

```

117     /// <para></para>
118     /// </param>
119     /// <param name="left">
120     /// <para>The left.</para>
121     /// <para></para>
122     /// </param>
123     [MethodImpl(MethodImplOptions.AggressiveInlining)]
124     protected override void SetLeft(TLink node, TLink left) =>
125         ↪ GetLinkIndexPartReference(node).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(TLink node, TLink right) =>
143         ↪ GetLinkIndexPartReference(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 TLink GetSize(TLink node) =>
161         ↪ GetLinkIndexPartReference(node).SizeAsSource;
162
163     /// <summary>
164     /// <para>
165     /// Sets the size using the specified node.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="node">
170     /// <para>The node.</para>
171     /// <para></para>
172     /// </param>
173     /// <param name="size">
174     /// <para>The size.</para>
175     /// <para></para>
176     /// </param>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override void SetSize(TLink node, TLink size) =>
179         ↪ GetLinkIndexPartReference(node).SizeAsSource = 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">
188     /// <para>The link.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The link</para>
193     /// <para></para>
194     /// </returns>

```



```

191 [MethodImpl(MethodImplOptions.AggressiveInlining)]
192 protected override TLink GetTreeRoot(TLink link) =>
    ↳ GetLinkIndexPartReference(link).RootAsSource;
193
194 /// <summary>
195 /// <para>
196 /// Gets the base part value using the specified link.
197 /// </para>
198 /// <para></para>
199 /// </summary>
200 /// <param name="link">
201 /// <para>The link.</para>
202 /// <para></para>
203 /// </param>
204 /// <returns>
205 /// <para>The link</para>
206 /// <para></para>
207 /// </returns>
208 [MethodImpl(MethodImplOptions.AggressiveInlining)]
209 protected override TLink GetBasePartValue(TLink link) =>
    ↳ GetLinkDataPartReference(link).Source;
210
211 /// <summary>
212 /// <para>
213 /// Gets the key part value using the specified link.
214 /// </para>
215 /// <para></para>
216 /// </summary>
217 /// <param name="link">
218 /// <para>The link.</para>
219 /// <para></para>
220 /// </param>
221 /// <returns>
222 /// <para>The link</para>
223 /// <para></para>
224 /// </returns>
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 protected override TLink GetKeyPartValue(TLink link) =>
    ↳ GetLinkDataPartReference(link).Target;
227
228 /// <summary>
229 /// <para>
230 /// Clears the node using the specified node.
231 /// </para>
232 /// <para></para>
233 /// </summary>
234 /// <param name="node">
235 /// <para>The node.</para>
236 /// <para></para>
237 /// </param>
238 [MethodImpl(MethodImplOptions.AggressiveInlining)]
239 protected override void ClearNode(TLink 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 TLink Search(TLink source, TLink target) =>
266             ↪ SearchCore(GetTreeRoot(source), target);
267     }
}

```

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

```

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

```

```

69     /// <para>The ref link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref TLink GetRightReference(TLink 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 TLink GetLeft(TLink 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 TLink GetRight(TLink 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(TLink node, TLink left) =>
128        ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">
141    /// <para>The right.</para>
142    /// <para></para>
143    /// </param>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected override void SetRight(TLink node, TLink right) =>
146        ↪ GetLinkIndexPartReference(node).RightAsSource = right;

```

```

142     /// <summary>
143     /// <para>
144     /// Gets the size using the specified node.
145     /// </para>
146     /// <para></para>
147     /// </summary>
148     /// <param name="node">
149     /// <para>The node.</para>
150     /// <para></para>
151     /// </param>
152     /// <returns>
153     /// <para>The link</para>
154     /// <para></para>
155     /// </returns>
156     [MethodImpl(MethodImplOptions.AggressiveInlining)]
157     protected override TLink GetSize(TLink node) =>
158         ↪ GetLinkIndexPartReference(node).SizeAsSource;
159
160     /// <summary>
161     /// <para>
162     /// Sets the size using the specified node.
163     /// </para>
164     /// <para></para>
165     /// </summary>
166     /// <param name="node">
167     /// <para>The node.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="size">
171     /// <para>The size.</para>
172     /// <para></para>
173     /// </param>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     protected override void SetSize(TLink node, TLink 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 TLink GetTreeRoot(TLink 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 TLink GetBasePartValue(TLink 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>

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.Split.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the internal links targets recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14    public unsafe class InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
15        ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↪ cref="InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <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 InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
    ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
    ↪ base(constants, linksDataParts, linksIndexParts, header) { }

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

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

74
75    /// <summary>
76    /// <para>
77    /// Gets the left using the specified node.
78    /// </para>
79    /// <para></para>
80    /// </summary>
81    /// <param name="node">
82    /// <para>The node.</para>
83    /// <para></para>
84    /// </param>
85    /// <returns>
86    /// <para>The link</para>
87    /// <para></para>
88    /// </returns>
89    [MethodImpl(MethodImplOptions.AggressiveInlining)]
90    protected override TLink GetLeft(TLink node) =>
    ↪ GetLinkIndexPartReference(node).LeftAsTarget;

91
92    /// <summary>

```

```

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

```

```

167    /// <para>The node.</para>
168    /// <para></para>
169    /// </param>
170    /// <param name="size">
171    /// <para>The size.</para>
172    /// <para></para>
173    /// </param>
174    [MethodImpl(MethodImplOptions.AggressiveInlining)]
175    protected override void SetSize(TLink node, TLink size) =>
176        ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
177
178    /// <summary>
179    /// <para>
180    /// Gets the tree root 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 TLink GetTreeRoot(TLink link) =>
194        ↪ GetLinkIndexPartReference(link).RootAsTarget;
195
196    /// <summary>
197    /// <para>
198    /// Gets the base part value using the specified link.
199    /// </para>
200    /// <para></para>
201    /// </summary>
202    /// <param name="link">
203    /// <para>The link.</para>
204    /// <para></para>
205    /// </param>
206    /// <returns>
207    /// <para>The link</para>
208    /// <para></para>
209    /// </returns>
210    [MethodImpl(MethodImplOptions.AggressiveInlining)]
211    protected override TLink GetBasePartValue(TLink link) =>
212        ↪ GetLinkDataPartReference(link).Target;
213
214    /// <summary>
215    /// <para>
216    /// Gets the key part value using the specified link.
217    /// </para>
218    /// <para></para>
219    /// </summary>
220    /// <param name="link">
221    /// <para>The link.</para>
222    /// <para></para>
223    /// </param>
224    /// <returns>
225    /// <para>The link</para>
226    /// <para></para>
227    /// </returns>
228    [MethodImpl(MethodImplOptions.AggressiveInlining)]
229    protected override TLink GetKeyPartValue(TLink link) =>
230        ↪ GetLinkDataPartReference(link).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(TLink node)
244    {

```



```

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

```

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

```

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

```

```

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

```

```

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

```

```

193
194     /// <summary>
195     /// <para>
196     /// Gets the base part value using the specified link.
197     /// </para>
198     /// <para></para>
199     /// </summary>
200     /// <param name="link">
201     /// <para>The link.</para>
202     /// <para></para>
203     /// </param>
204     /// <returns>
205     /// <para>The link</para>
206     /// <para></para>
207     /// </returns>
208     [MethodImpl(MethodImplOptions.AggressiveInlining)]
209     protected override TLink GetBasePartValue(TLink link) =>
210         ↪ GetLinkDataPartReference(link).Target;
211
212     /// <summary>
213     /// <para>
214     /// Gets the key part value using the specified link.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="link">
219     /// <para>The link.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>The link</para>
224     /// <para></para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected override TLink GetKeyPartValue(TLink link) =>
228         ↪ GetLinkDataPartReference(link).Source;
229
230     /// <summary>
231     /// <para>
232     /// Clears the node using the specified node.
233     /// </para>
234     /// <para></para>
235     /// </summary>
236     /// <param name="node">
237     /// <para>The node.</para>
238     /// <para></para>
239     /// </param>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     protected override void ClearNode(TLink node)
242     {
243         ref var link = ref GetLinkIndexPartReference(node);
244         link.LeftAsTarget = Zero;
245         link.RightAsTarget = Zero;
246         link.SizeAsTarget = Zero;
247     }
248
249     /// <summary>
250     /// <para>
251     /// Searches the source.
252     /// </para>
253     /// <para></para>
254     /// </summary>
255     /// <param name="source">
256     /// <para>The source.</para>
257     /// <para></para>
258     /// </param>
259     /// <param name="target">
260     /// <para>The target.</para>
261     /// <para></para>
262     /// </param>
263     /// <returns>
264     /// <para>The link</para>
265     /// <para></para>
266     /// </returns>
267     public override TLink Search(TLink source, TLink target) =>
268         ↪ SearchCore(GetTreeRoot(target), source);
269 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using static System.Runtime.CompilerServices.Unsafe;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.Split.Generic
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the split memory links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
18     public unsafe class SplitMemoryLinks<TLink> : SplitMemoryLinksBase<TLink> where TLink :
19         ↳ struct
20     {
21         private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
24         private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
25         private byte* _header;
26         private byte* _linksDataParts;
27         private byte* _linksIndexParts;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="SplitMemoryLinks"/> 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 SplitMemoryLinks(string dataMemory, string indexMemory) : this(new
45             ↳ FileMappedResizableDirectMemory(dataMemory), new
46             ↳ FileMappedResizableDirectMemory(indexMemory)) { }
47
48         /// <summary>
49         /// <para>
50         /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="dataMemory">
55         /// <para>A data memory.</para>
56         /// <para></para>
57         /// </param>
58         /// <param name="indexMemory">
59         /// <para>A index memory.</para>
60         /// <para></para>
61         /// </param>
62         [MethodImpl(MethodImplOptions.AggressiveInlining)]
63         public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
64             ↳ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
65
66         /// <summary>
67         /// <para>
68         /// Initializes a new <see cref="SplitMemoryLinks"/> 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>

```

```

75     /// </param>
76     /// <param name="memoryReservationStep">
77     /// <para>A memory reservation step.</para>
78     /// <para></para>
79     /// </param>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↳ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
        ↳ IndexTreeType.Default, useLinkedList: true) { }

82
83     /// <summary>
84     /// <para>
85     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <param name="dataMemory">
90     /// <para>A data memory.</para>
91     /// <para></para>
92     /// </param>
93     /// <param name="indexMemory">
94     /// <para>A index memory.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="memoryReservationStep">
98     /// <para>A memory reservation step.</para>
99     /// <para></para>
100    /// </param>
101    /// <param name="constants">
102    /// <para>A constants.</para>
103    /// <para></para>
104    /// </param>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
        ↳ this(dataMemory, indexMemory, memoryReservationStep, constants,
        ↳ IndexTreeType.Default, useLinkedList: true) { }

107
108    /// <summary>
109    /// <para>
110    /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
111    /// </para>
112    /// <para></para>
113    /// </summary>
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<TLink> constants,
        ↳ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↳ memoryReservationStep, constants, useLinkedList)
140    {
141        if (indexTreeType == IndexTreeType.SizeBalancedTree)
142        {

```

```

143     _createInternalSourceTreeMethods = () => new
144         ↪ InternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
145         ↪ _linksDataParts, _linksIndexParts, _header);
146     _createExternalSourceTreeMethods = () => new
147         ↪ ExternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
148         ↪ _linksDataParts, _linksIndexParts, _header);
149     _createInternalTargetTreeMethods = () => new
150         ↪ InternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
151         ↪ _linksDataParts, _linksIndexParts, _header);
152     _createExternalTargetTreeMethods = () => new
153         ↪ ExternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
154         ↪ _linksDataParts, _linksIndexParts, _header);
155 }
156 else
157 {
158     _createInternalSourceTreeMethods = () => new
159         ↪ InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
160         ↪ _linksDataParts, _linksIndexParts, _header);
161     _createExternalSourceTreeMethods = () => new
162         ↪ ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
163         ↪ _linksDataParts, _linksIndexParts, _header);
164     _createInternalTargetTreeMethods = () => new
165         ↪ InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
166         ↪ _linksDataParts, _linksIndexParts, _header);
167     _createExternalTargetTreeMethods = () => new
168         ↪ ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
169         ↪ _linksDataParts, _linksIndexParts, _header);
170 }
171 Init(dataMemory, indexMemory);
172 }
173
174 /// <summary>
175 /// <para>
176 /// Sets the pointers using the specified data memory.
177 /// </para>
178 /// <para></para>
179 /// </summary>
180 /// <param name="dataMemory">
181 /// <para>The data memory.</para>
182 /// </param>
183 /// <param name="indexMemory">
184 /// <para>The index memory.</para>
185 /// </param>
186 [MethodImpl(MethodImplOptions.AggressiveInlining)]
187 protected override void SetPointers(IResizableDirectMemory dataMemory,
188     ↪ IResizableDirectMemory indexMemory)
189 {
190     _linksDataParts = (byte*)dataMemory.Pointer;
191     _linksIndexParts = (byte*)indexMemory.Pointer;
192     _header = _linksIndexParts;
193     if (_useLinkedList)
194     {
195         InternalSourcesListMethods = new
196             ↪ InternalLinksSourcesLinkedListMethods<TLink>(Constants, _linksDataParts,
197             ↪ _linksIndexParts);
198     }
199     else
200     {
201         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
202     }
203     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
204     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
205     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
206     UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_linksDataParts, _header);
207 }
208
209 /// <summary>
210 /// <para>
211 /// Resets the pointers.
212 /// </para>
213 /// <para></para>
214 /// </summary>
215 [MethodImpl(MethodImplOptions.AggressiveInlining)]
216 protected override void ResetPointers()
217 {

```

```

201         base.ResetPointers();
202         _linksDataParts = null;
203         _linksIndexParts = null;
204         _header = null;
205     }
206
207     /// <summary>
208     /// <para>
209     /// Gets the header reference.
210     /// </para>
211     /// <para></para>
212     /// </summary>
213     /// <returns>
214     /// <para>A ref links header of t link</para>
215     /// <para></para>
216     /// </returns>
217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
218     protected override ref LinksHeader<TLink> GetHeaderReference() => ref
219         ↪ AsRef<LinksHeader<TLink>>(_header);
220
221     /// <summary>
222     /// <para>
223     /// Gets the link data part reference using the specified link index.
224     /// </para>
225     /// <para></para>
226     /// </summary>
227     /// <param name="linkIndex">
228     /// <para>The link index.</para>
229     /// <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<TLink> GetLinkDataPartReference(TLink linkIndex)
237         ↪ => ref AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (LinkDataPartSizeInBytes *
238         ↪ ConvertToInt64(linkIndex)));
239
240     /// <summary>
241     /// <para>
242     /// Gets the link index part reference using the specified link index.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="linkIndex">
247     /// <para>The link index.</para>
248     /// <para></para>
249     /// </param>
250     /// <returns>
251     /// <para>A ref raw link index part of t link</para>
252     /// <para></para>
253     /// </returns>
254     [MethodImpl(MethodImplOptions.AggressiveInlining)]
255     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
256         ↪ linkIndex) => ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
257         ↪ (LinkIndexPartSizeInBytes * ConvertToInt64(linkIndex)));
258 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Disposables;
5  using Platform.Singletons;
6  using Platform.Converters;
7  using Platform.Numbers;
8  using Platform.Memory;
9  using Platform.Data.Exceptions;
10 using Platform.Delegates;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Memory.Split.Generic
15 {
16     /// <summary>
17     /// <para>
18     /// Represents the split memory links base.

```



```

19  /// </para>
20  /// <para></para>
21  /// </summary>
22  /// <seealso cref="DisposableBase"/>
23  /// <seealso cref="ILinks{TLink}"/>
24  public abstract class SplitMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink> where
    ↳ TLink : struct
25  {
26      private static readonly EqualityComparer<TLink> _equalityComparer =
        ↳ EqualityComparer<TLink>.Default;
27      private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
28      private static readonly uncheckedConverter<TLink, long> _addressToInt64Converter =
        ↳ uncheckedConverter<TLink, long>.Default;
29      private static readonly uncheckedConverter<long, TLink> _int64ToAddressConverter =
        ↳ uncheckedConverter<long, TLink>.Default;
30      private static readonly TLink _zero = default;
31      private static readonly TLink _one = Arithmetic.Increment(_zero);
32
33      /// <summary>Возвращает размер одной связи в байтах.</summary>
34      /// <remarks>
35      /// Используется только во вне класса, не рекомендуется использовать внутри.
36      /// Так как во вне не обязательно будет доступен unsafe C#.
37      /// </remarks>
38      public static readonly long LinkDataPartSizeInBytes = RawLinkDataPart<TLink>.SizeInBytes;
39
40      /// <summary>
41      /// <para>
42      /// The size in bytes.
43      /// </para>
44      /// <para></para>
45      /// </summary>
46      public static readonly long LinkIndexPartSizeInBytes =
        ↳ RawLinkIndexPart<TLink>.SizeInBytes;
47
48      /// <summary>
49      /// <para>
50      /// The size in bytes.
51      /// </para>
52      /// <para></para>
53      /// </summary>
54      public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
55
56      /// <summary>
57      /// <para>
58      /// The default links size step.
59      /// </para>
60      /// <para></para>
61      /// </summary>
62      public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
63
64      /// <summary>
65      /// <para>
66      /// The data memory.
67      /// </para>
68      /// <para></para>
69      /// </summary>
70      protected readonly IResizableDirectMemory _dataMemory;
71      /// <summary>
72      /// <para>
73      /// The index memory.
74      /// </para>
75      /// <para></para>
76      /// </summary>
77      protected readonly IResizableDirectMemory _indexMemory;
78      /// <summary>
79      /// <para>
80      /// The use linked list.
81      /// </para>
82      /// <para></para>
83      /// </summary>
84      protected readonly bool _useLinkedList;
85      /// <summary>
86      /// <para>
87      /// The data memory reservation step in bytes.
88      /// </para>
89      /// <para></para>
90      /// </summary>
91      protected readonly long _dataMemoryReservationStepInBytes;
92      /// <summary>

```

```

93     /// <para>
94     /// The index memory reservation step in bytes.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     protected readonly long _indexMemoryReservationStepInBytes;
99
100    /// <summary>
101    /// <para>
102    /// The internal sources list methods.
103    /// </para>
104    /// <para></para>
105    /// </summary>
106    protected InternalLinksSourcesLinkedListMethods<TLink> InternalSourcesListMethods;
107    /// <summary>
108    /// <para>
109    /// The internal sources tree methods.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    protected ILinksTreeMethods<TLink> InternalSourcesTreeMethods;
114    /// <summary>
115    /// <para>
116    /// The external sources tree methods.
117    /// </para>
118    /// <para></para>
119    /// </summary>
120    protected ILinksTreeMethods<TLink> ExternalSourcesTreeMethods;
121    /// <summary>
122    /// <para>
123    /// The internal targets tree methods.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    protected ILinksTreeMethods<TLink> InternalTargetsTreeMethods;
128    /// <summary>
129    /// <para>
130    /// The external targets tree methods.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    protected ILinksTreeMethods<TLink> ExternalTargetsTreeMethods;
135    // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
136    // ↳ нужно использовать не список а дерево, так как так можно быстрее проверить на
137    // ↳ наличие связи внутри
138    /// <summary>
139    /// <para>
140    /// The unused links list methods.
141    /// </para>
142    /// <para></para>
143    /// </summary>
144    protected ILinksListMethods<TLink> UnusedLinksListMethods;
145
146    /// <summary>
147    /// Возвращает общее число связей находящихся в хранилище.
148    /// </summary>
149    protected virtual TLink Total
150    {
151        [MethodImpl(MethodImplOptions.AggressiveInlining)]
152        get
153        {
154            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<TLink> Constants
166    {
167        [MethodImpl(MethodImplOptions.AggressiveInlining)]
168        get;
169    }
170
171    /// <summary>

```

```

170    /// <para>
171    /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
172    /// </para>
173    /// </summary>
174    /// <param name="dataMemory">
175    /// <para>A data memory.</para>
176    /// </param>
177    /// <param name="indexMemory">
178    /// <para>A index memory.</para>
179    /// </param>
180    /// <param name="memoryReservationStep">
181    /// <para>A memory reservation step.</para>
182    /// </param>
183    /// <param name="constants">
184    /// <para>A constants.</para>
185    /// </param>
186    /// <param name="useLinkedList">
187    /// <para>A use linked list.</para>
188    /// </param>
189    [MethodImpl(MethodImplOptions.AggressiveInlining)]
190    protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants, bool
    ↪ useLinkedList)
191    {
192        _dataMemory = dataMemory;
193        _indexMemory = indexMemory;
194        _dataMemoryReservationStepInBytes = memoryReservationStep * LinkDataPartSizeInBytes;
195        _indexMemoryReservationStepInBytes = memoryReservationStep *
    ↪ LinkIndexPartSizeInBytes;
196        _useLinkedList = useLinkedList;
197        Constants = constants;
198    }
199
200    /// <summary>
201    /// <para>
202    /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
203    /// </para>
204    /// </summary>
205    /// <param name="dataMemory">
206    /// <para>A data memory.</para>
207    /// </param>
208    /// <param name="indexMemory">
209    /// <para>A index memory.</para>
210    /// </param>
211    /// <param name="memoryReservationStep">
212    /// <para>A memory reservation step.</para>
213    /// </param>
214    [MethodImpl(MethodImplOptions.AggressiveInlining)]
215    protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
    ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance, useLinkedList: true)
216    { }
217
218    /// <summary>
219    /// <para>
220    /// Inits the data memory.
221    /// </para>
222    /// </summary>
223    /// <param name="dataMemory">
224    /// <para>The data memory.</para>
225    /// </param>
226    /// <param name="indexMemory">
227    /// <para>The index memory.</para>
228    /// </param>
229    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

242 protected virtual void Init(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory)
243 {
244     // Read allocated links from header
245     if (indexMemory.ReservedCapacity < LinkHeaderSizeInBytes)
246     {
247         indexMemory.ReservedCapacity = LinkHeaderSizeInBytes;
248     }
249     SetPointers(dataMemory, indexMemory);
250     ref var header = ref GetHeaderReference();
251     var allocatedLinks = ConvertToInt64(header.AllocatedLinks);
252     // Adjust reserved capacity
253     var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
254     if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
255     {
256         minimumDataReservedCapacity = dataMemory.UsedCapacity;
257     }
258     if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
259     {
260         minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
261     }
262     var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
263     if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
264     {
265         minimumIndexReservedCapacity = indexMemory.UsedCapacity;
266     }
267     if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
268     {
269         minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
270     }
271     // Check for alignment
272     if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
273     {
274         minimumDataReservedCapacity = ((minimumDataReservedCapacity /
    ↪ _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
    ↪ _dataMemoryReservationStepInBytes;
275     }
276     if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)
277     {
278         minimumIndexReservedCapacity = ((minimumIndexReservedCapacity /
    ↪ _indexMemoryReservationStepInBytes) * _indexMemoryReservationStepInBytes) +
    ↪ _indexMemoryReservationStepInBytes;
279     }
280     if (dataMemory.ReservedCapacity != minimumDataReservedCapacity)
281     {
282         dataMemory.ReservedCapacity = minimumDataReservedCapacity;
283     }
284     if (indexMemory.ReservedCapacity != minimumIndexReservedCapacity)
285     {
286         indexMemory.ReservedCapacity = minimumIndexReservedCapacity;
287     }
288     SetPointers(dataMemory, indexMemory);
289     header = ref GetHeaderReference();
290     // Ensure correctness _memory.UsedCapacity over _header->AllocatedLinks
291     // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
292     dataMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
    ↪ LinkDataPartSizeInBytes) + LinkDataPartSizeInBytes; // First link is read only
    ↪ zero link.
293     indexMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
    ↪ LinkIndexPartSizeInBytes) + LinkHeaderSizeInBytes;
294     // Ensure correctness _memory.ReservedLinks over _header->ReservedCapacity
295     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
296     header.ReservedLinks = ConvertToAddress((dataMemory.ReservedCapacity -
    ↪ LinkDataPartSizeInBytes) / LinkDataPartSizeInBytes);
297 }
298
299 /// <summary>
300 /// <para>
301 /// Counts the substitution.
302 /// </para>
303 /// <para></para>
304 /// </summary>
305 /// <param name="restriction">
306 /// <para>The substitution.</para>
307 /// <para></para>
308 /// </param>
309 /// <exception cref="NotSupportedException">
310 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>

```

```

311 /// <para></para>
312 /// </exception>
313 /// <returns>
314 /// <para>The link</para>
315 /// <para></para>
316 /// </returns>
317 [MethodImpl(MethodImplOptions.AggressiveInlining)]
318 public virtual TLink Count(IList<TLink>? restriction)
319 {
320     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
321     if (restriction.Count == 0)
322     {
323         return Total;
324     }
325     var constants = Constants;
326     var any = constants.Any;
327     var index = restriction[constants.IndexPart];
328     if (restriction.Count == 1)
329     {
330         if (AreEqual(index, any))
331         {
332             return Total;
333         }
334         return Exists(index) ? GetOne() : GetZero();
335     }
336     if (restriction.Count == 2)
337     {
338         var value = restriction[1];
339         if (AreEqual(index, any))
340         {
341             if (AreEqual(value, any))
342             {
343                 return Total; // Any - как отсутствие ограничения
344             }
345             var externalReferencesRange = constants.ExternalReferencesRange;
346             if (externalReferencesRange.HasValue &&
347                 ↪ externalReferencesRange.Value.Contains(value))
348             {
349                 return Add(ExternalSourcesTreeMethods.CountUsages(value),
350                     ↪ ExternalTargetsTreeMethods.CountUsages(value));
351             }
352             else
353             {
354                 if (_useLinkedList)
355                 {
356                     return Add(InternalSourcesListMethods.CountUsages(value),
357                         ↪ InternalTargetsTreeMethods.CountUsages(value));
358                 }
359                 else
360                 {
361                     return Add(InternalSourcesTreeMethods.CountUsages(value),
362                         ↪ InternalTargetsTreeMethods.CountUsages(value));
363                 }
364             }
365         }
366         else
367         {
368             if (!Exists(index))
369             {
370                 return GetZero();
371             }
372             if (AreEqual(value, any))
373             {
374                 return GetOne();
375             }
376             ref var storedLinkValue = ref GetLinkDataPartReference(index);
377             if (AreEqual(storedLinkValue.Source, value) ||
378                 ↪ AreEqual(storedLinkValue.Target, value))
379             {
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     }
390 }

```

```

384 var target = restriction[constants.TargetPart];
385 if (AreEqual(index, any))
386 {
387     if (AreEqual(source, any) && AreEqual(target, any))
388     {
389         return Total;
390     }
391     else if (AreEqual(source, any))
392     {
393         if (externalReferencesRange.HasValue &&
394             ↪ externalReferencesRange.Value.Contains(target))
395         {
396             return ExternalTargetsTreeMethods.CountUsages(target);
397         }
398         else
399         {
400             return InternalTargetsTreeMethods.CountUsages(target);
401         }
402     }
403     else if (AreEqual(target, any))
404     {
405         if (externalReferencesRange.HasValue &&
406             ↪ externalReferencesRange.Value.Contains(source))
407         {
408             return ExternalSourcesTreeMethods.CountUsages(source);
409         }
410         else
411         {
412             if (_useLinkedList)
413             {
414                 return InternalSourcesListMethods.CountUsages(source);
415             }
416             else
417             {
418                 return InternalSourcesTreeMethods.CountUsages(source);
419             }
420         }
421     }
422     else //if(source != Any && target != Any)
423     {
424         // ЭКВИВАЛЕНТ Exists(source, target) => Count(Any, source, target) > 0
425         TLink link;
426         if (externalReferencesRange.HasValue)
427         {
428             if (externalReferencesRange.Value.Contains(source) &&
429                 ↪ externalReferencesRange.Value.Contains(target))
430             {
431                 link = ExternalSourcesTreeMethods.Search(source, target);
432             }
433             else if (externalReferencesRange.Value.Contains(source))
434             {
435                 link = InternalTargetsTreeMethods.Search(source, target);
436             }
437             else if (externalReferencesRange.Value.Contains(target))
438             {
439                 if (_useLinkedList)
440                 {
441                     link = ExternalSourcesTreeMethods.Search(source, target);
442                 }
443                 else
444                 {
445                     link = InternalSourcesTreeMethods.Search(source, target);
446                 }
447             }
448             else
449             {
450                 if (_useLinkedList ||
451                     ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
452                     ↪ InternalTargetsTreeMethods.CountUsages(target)))
453                 {
454                     link = InternalTargetsTreeMethods.Search(source, target);
455                 }
456                 else
457                 {
458                     link = InternalSourcesTreeMethods.Search(source, target);
459                 }
460             }
461         }
462     }
463 }

```

```

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

```

```

530 /// </returns>
531 [MethodImpl(MethodImplOptions.AggressiveInlining)]
532 public virtual TLink Each(ICollection<TLink>? restriction, ReadHandler<TLink>? handler)
533 {
534     var constants = Constants;
535     var @break = constants.Break;
536     if (restriction.Count == 0)
537     {
538         for (var link = GetOne(); LessOrEqualThan(link,
539             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
540         {
541             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
542             {
543                 return @break;
544             }
545         }
546         return @break;
547     }
548     var @continue = constants.Continue;
549     var any = constants.Any;
550     var index = restriction[constants.IndexPart];
551     if (restriction.Count == 1)
552     {
553         if (AreEqual(index, any))
554         {
555             return Each(Array.Empty<TLink>(), handler);
556         }
557         if (!Exists(index))
558         {
559             return @continue;
560         }
561         return handler(GetLinkStruct(index));
562     }
563     if (restriction.Count == 2)
564     {
565         var value = restriction[1];
566         if (AreEqual(index, any))
567         {
568             if (AreEqual(value, any))
569             {
570                 return Each(Array.Empty<TLink>(), handler);
571             }
572             if (AreEqual(Each(new Link<TLink>(index, value, any), handler), @break))
573             {
574                 return @break;
575             }
576             return Each(new Link<TLink>(index, any, value), handler);
577         }
578         else
579         {
580             if (!Exists(index))
581             {
582                 return @continue;
583             }
584             if (AreEqual(value, any))
585             {
586                 return handler(GetLinkStruct(index));
587             }
588             ref var storedLinkValue = ref GetLinkDataPartReference(index);
589             if (AreEqual(storedLinkValue.Source, value) ||
590                 AreEqual(storedLinkValue.Target, value))
591             {
592                 return handler(GetLinkStruct(index));
593             }
594             return @continue;
595         }
596     }
597     if (restriction.Count == 3)
598     {
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<TLink>(), handler);
607             }
608         }
609     }
610 }

```



```

607     else if (AreEqual(source, any))
608     {
609         if (externalReferencesRange.HasValue &&
610             ↪ externalReferencesRange.Value.Contains(target))
611         {
612             return ExternalTargetsTreeMethods.EachUsage(target, handler);
613         }
614         else
615         {
616             return InternalTargetsTreeMethods.EachUsage(target, handler);
617         }
618     }
619     else if (AreEqual(target, any))
620     {
621         if (externalReferencesRange.HasValue &&
622             ↪ externalReferencesRange.Value.Contains(source))
623         {
624             return ExternalSourcesTreeMethods.EachUsage(source, handler);
625         }
626         else
627         {
628             if (_useLinkedList)
629             {
630                 return InternalSourcesListMethods.EachUsage(source, handler);
631             }
632             else
633             {
634                 return InternalSourcesTreeMethods.EachUsage(source, handler);
635             }
636         }
637     }
638     else //if(source != Any && target != Any)
639     {
640         TLink link;
641         if (externalReferencesRange.HasValue)
642         {
643             if (externalReferencesRange.Value.Contains(source) &&
644                 ↪ externalReferencesRange.Value.Contains(target))
645             {
646                 link = ExternalSourcesTreeMethods.Search(source, target);
647             }
648             else if (externalReferencesRange.Value.Contains(source))
649             {
650                 link = InternalTargetsTreeMethods.Search(source, target);
651             }
652             else if (externalReferencesRange.Value.Contains(target))
653             {
654                 if (_useLinkedList)
655                 {
656                     link = ExternalSourcesTreeMethods.Search(source, target);
657                 }
658                 else
659                 {
660                     link = InternalSourcesTreeMethods.Search(source, target);
661                 }
662             }
663             else
664             {
665                 if (_useLinkedList ||
666                     ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
667                     ↪ InternalTargetsTreeMethods.CountUsages(target)))
668                 {
669                     link = InternalTargetsTreeMethods.Search(source, target);
670                 }
671                 else
672                 {
673                     link = InternalSourcesTreeMethods.Search(source, target);
674                 }
675             }
676         }
677         else
678         {
679             if (_useLinkedList ||
680                 ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
681                 ↪ InternalTargetsTreeMethods.CountUsages(target)))
682             {
683                 link = InternalTargetsTreeMethods.Search(source, target);
684             }
685             else
686             {
687                 link = InternalSourcesTreeMethods.Search(source, target);
688             }
689         }
690     }
691 }

```

```

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(TLink);
707     if (AreEqual(source, any))
708     {
709         value = target;
710     }
711     if (AreEqual(target, any))
712     {
713         value = source;
714     }
715     if (AreEqual(storedLinkValue.Source, value) ||
716         AreEqual(storedLinkValue.Target, value))
717     {
718         return handler(GetLinkStruct(index));
719     }
720     return @continue;
721 }
722 }
723 throw new NotSupportedException("Другие размеры и способы ограничений не
        ↳ поддерживаются.");
724 }
725
726 /// <remarks>
727 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
728 ↳ в другом месте (но не в менеджере памяти, а в логике Links)
729 /// </remarks>
730 [MethodImpl(MethodImplOptions.AggressiveInlining)]
731 public virtual TLink Update(ICollection<TLink>? restriction, ICollection<TLink>? substitution,
732     ↳ WriteHandler<TLink>? 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     }
755 }

```

```

750     else
751     {
752         if (_useLinkedList)
753         {
754             InternalSourcesListMethods.Detach(source, linkIndex);
755         }
756         else
757         {
758             InternalSourcesTreeMethods.Detach(ref
759                 ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
760         }
761     }
762     if (!AreEqual(target, @null))
763     {
764         if (externalReferencesRange.HasValue &&
765             ↪ externalReferencesRange.Value.Contains(target))
766         {
767             ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
768         }
769         else
770         {
771             InternalTargetsTreeMethods.Detach(ref
772                 ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
773         }
774     }
775     source = link.Source = substitution[constants.SourcePart];
776     target = link.Target = substitution[constants.TargetPart];
777     if (!AreEqual(source, @null))
778     {
779         if (externalReferencesRange.HasValue &&
780             ↪ externalReferencesRange.Value.Contains(source))
781         {
782             ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
783         }
784         else
785         {
786             if (_useLinkedList)
787             {
788                 InternalSourcesListMethods.AttachAsLast(source, linkIndex);
789             }
790             else
791             {
792                 InternalSourcesTreeMethods.Attach(ref
793                     ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
794             }
795         }
796     }
797     if (!AreEqual(target, @null))
798     {
799         if (externalReferencesRange.HasValue &&
800             ↪ externalReferencesRange.Value.Contains(target))
801         {
802             ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
803         }
804         else
805         {
806             InternalTargetsTreeMethods.Attach(ref
807                 ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
808         }
809     }
810     return handler?.Invoke(before, new Link<TLink>(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 TLink Create(ICollection<TLink>? substitution, WriteHandler<TLink>? handler)
820 {
821     ref var header = ref GetHeaderReference();
822     var freeLink = header.FirstFreeLink;
823     if (!AreEqual(freeLink, Constants.Null))
824     {
825         UnusedLinksListMethods.Detach(freeLink);
826     }
827 }

```

```

819     else
820     {
821         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
822         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
823         {
824             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
825         }
826         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
827         {
828             _dataMemory.ReservedCapacity += _dataMemory.ReservationStepInBytes;
829             _indexMemory.ReservedCapacity += _indexMemory.ReservationStepInBytes;
830             SetPointers(_dataMemory, _indexMemory);
831             header = ref GetHeaderReference();
832             header.ReservedLinks = ConvertToAddress(_dataMemory.ReservedCapacity /
833                 ↪ LinkDataPartSizeInBytes);
834         }
835         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
836         _dataMemory.UsedCapacity += LinkDataPartSizeInBytes;
837         _indexMemory.UsedCapacity += LinkIndexPartSizeInBytes;
838     }
839     return handler?.Invoke(null, GetLinkStruct(freeLink)) ?? Constants.Continue;
840 }
841
842 /// <summary>
843 /// <para>
844 /// Deletes the substitution.
845 /// </para>
846 /// </summary>
847 /// <param name="restriction">
848 /// <para>The substitution.</para>
849 /// </param>
850 [MethodImpl(MethodImplOptions.AggressiveInlining)]
851 public virtual TLink Delete(IList<TLink>? restriction, WriteHandler<TLink>? handler)
852 {
853     ref var header = ref GetHeaderReference();
854     var link = restriction[Constants.IndexPart];
855     var before = GetLinkStruct(link);
856     if (LessThan(link, header.AllocatedLinks))
857     {
858         UnusedLinksListMethods.AttachAsFirst(link);
859     }
860     else if (AreEqual(link, header.AllocatedLinks))
861     {
862         header.AllocatedLinks = Decrement(header.AllocatedLinks);
863         _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
864         _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
865         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
866         //   ↪ пока не дойдём до первой существующей связи
867         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
868         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
869             ↪ 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 /// </summary>
885 /// <param name="linkIndex">
886 /// <para>The link index.</para>
887 /// </param>
888 /// <returns>
889 /// <para>A list of t link</para>
890 /// </returns>
891 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

894 public IList<TLink>? GetLinkStruct(TLink linkIndex)
895 {
896     ref var link = ref GetLinkDataPartReference(linkIndex);
897     return new Link<TLink>(linkIndex, link.Source, link.Target);
898 }
899
900 /// <remarks>
901 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
902   → адрес реально поменялся
903 ///
904 /// Указатель this.links может быть в том же месте,
905 /// так как 0-я связь не используется и имеет такой же размер как Header,
906 /// поэтому header размещается в том же месте, что и 0-я связь
907 /// </remarks>
908 [MethodImpl(MethodImplOptions.AggressiveInlining)]
909 protected abstract void SetPointers(IResizableDirectMemory dataMemory,
910   → IResizableDirectMemory indexMemory);
911
912 /// <summary>
913 /// <para>
914 /// Resets the pointers.
915 /// </para>
916 /// <para></para>
917 /// </summary>
918 [MethodImpl(MethodImplOptions.AggressiveInlining)]
919 protected virtual void ResetPointers()
920 {
921     InternalSourcesListMethods = null;
922     InternalSourcesTreeMethods = null;
923     ExternalSourcesTreeMethods = null;
924     InternalTargetsTreeMethods = null;
925     ExternalTargetsTreeMethods = null;
926     UnusedLinksListMethods = null;
927 }
928
929 /// <summary>
930 /// <para>
931 /// Gets the header reference.
932 /// </para>
933 /// <para></para>
934 /// </summary>
935 /// <returns>
936 /// <para>A ref links header of t link</para>
937 /// <para></para>
938 /// </returns>
939 [MethodImpl(MethodImplOptions.AggressiveInlining)]
940 protected abstract ref LinksHeader<TLink> GetHeaderReference();
941
942 /// <summary>
943 /// <para>
944 /// Gets the link data part reference using the specified link index.
945 /// </para>
946 /// <para></para>
947 /// </summary>
948 /// <param name="linkIndex">
949 /// <para>The link index.</para>
950 /// <para></para>
951 /// </param>
952 /// <returns>
953 /// <para>A ref raw link data part of t link</para>
954 /// <para></para>
955 /// </returns>
956 [MethodImpl(MethodImplOptions.AggressiveInlining)]
957 protected abstract ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex);
958
959 /// <summary>
960 /// <para>
961 /// Gets the link index part reference using the specified link index.
962 /// </para>
963 /// <para></para>
964 /// </summary>
965 /// <param name="linkIndex">
966 /// <para>The link index.</para>
967 /// <para></para>
968 /// </param>
969 /// <returns>
970 /// <para>A ref raw link index part of t link</para>
971 /// <para></para>
972 /// </returns>

```

```

970     /// </returns>
971     [MethodImpl(MethodImplOptions.AggressiveInlining)]
972     protected abstract ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
    ↪ linkIndex);

973
974     /// <summary>
975     /// <para>
976     /// Determines whether this instance exists.
977     /// </para>
978     /// <para></para>
979     /// </summary>
980     /// <param name="link">
981     /// <para>The link.</para>
982     /// <para></para>
983     /// </param>
984     /// <returns>
985     /// <para>The bool</para>
986     /// <para></para>
987     /// </returns>
988     [MethodImpl(MethodImplOptions.AggressiveInlining)]
989     protected virtual bool Exists(TLink 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(TLink 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    /// <summary>
1027    /// <para>
1028    /// Gets the one.
1029    /// </para>
1030    /// <para></para>
1031    /// </summary>
1032    /// <returns>
1033    /// <para>The link</para>
1034    /// <para></para>
1035    /// </returns>
1036    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1037    protected virtual TLink GetOne() => _one;

1038
1039    /// <summary>
1040    /// <para>
1041    /// Gets the zero.
1042    /// </para>
1043    /// <para></para>
1044    /// </summary>
1045    /// <returns>

```

```

1044    /// <para>The link</para>
1045    /// <para></para>
1046    /// </returns>
1047    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1048    protected virtual TLink GetZero() => default;
1049
1050    /// <summary>
1051    /// <para>
1052    /// Determines whether this instance are equal.
1053    /// </para>
1054    /// <para></para>
1055    /// </summary>
1056    /// <param name="first">
1057    /// <para>The first.</para>
1058    /// <para></para>
1059    /// </param>
1060    /// <param name="second">
1061    /// <para>The second.</para>
1062    /// <para></para>
1063    /// </param>
1064    /// <returns>
1065    /// <para>The bool</para>
1066    /// <para></para>
1067    /// </returns>
1068    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1069    protected virtual bool AreEqual(TLink first, TLink second) =>
1070        ↪ _equalityComparer.Equals(first, second);
1071
1072    /// <summary>
1073    /// <para>
1074    /// Determines whether this instance less than.
1075    /// </para>
1076    /// <para></para>
1077    /// </summary>
1078    /// <param name="first">
1079    /// <para>The first.</para>
1080    /// <para></para>
1081    /// </param>
1082    /// <param name="second">
1083    /// <para>The second.</para>
1084    /// <para></para>
1085    /// </param>
1086    /// <returns>
1087    /// <para>The bool</para>
1088    /// <para></para>
1089    /// </returns>
1090    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1091    protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
1092        ↪ second) < 0;
1093
1094    /// <summary>
1095    /// <para>
1096    /// Determines whether this instance less or equal than.
1097    /// </para>
1098    /// <para></para>
1099    /// </summary>
1100    /// <param name="first">
1101    /// <para>The first.</para>
1102    /// <para></para>
1103    /// </param>
1104    /// <param name="second">
1105    /// <para>The second.</para>
1106    /// <para></para>
1107    /// </param>
1108    /// <returns>
1109    /// <para>The bool</para>
1110    /// <para></para>
1111    /// </returns>
1112    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1113    protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
1114        ↪ _comparer.Compare(first, second) <= 0;
1115
1116    /// <summary>
1117    /// <para>
1118    /// Determines whether this instance greater than.
1119    /// </para>
1120    /// <para></para>
1121    /// </summary>

```

```

1119     /// <param name="first">
1120     /// <para>The first.</para>
1121     /// <para></para>
1122     /// </param>
1123     /// <param name="second">
1124     /// <para>The second.</para>
1125     /// <para></para>
1126     /// </param>
1127     /// <returns>
1128     /// <para>The bool</para>
1129     /// <para></para>
1130     /// </returns>
1131     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1132     protected virtual bool GreaterThan(TLink first, TLink second) =>
1133         ↪ _comparer.Compare(first, second) > 0;
1134
1135     /// <summary>
1136     /// <para>
1137     /// Determines whether this instance greater or equal than.
1138     /// </para>
1139     /// <para></para>
1140     /// </summary>
1141     /// <param name="first">
1142     /// <para>The first.</para>
1143     /// <para></para>
1144     /// </param>
1145     /// <param name="second">
1146     /// <para>The second.</para>
1147     /// <para></para>
1148     /// </param>
1149     /// <returns>
1150     /// <para>The bool</para>
1151     /// <para></para>
1152     /// </returns>
1153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1154     protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
1155         ↪ _comparer.Compare(first, second) >= 0;
1156
1157     /// <summary>
1158     /// <para>
1159     /// Converts the to int 64 using the specified value.
1160     /// </para>
1161     /// <para></para>
1162     /// </summary>
1163     /// <param name="value">
1164     /// <para>The value.</para>
1165     /// <para></para>
1166     /// </param>
1167     /// <returns>
1168     /// <para>The long</para>
1169     /// <para></para>
1170     /// </returns>
1171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1172     protected virtual long ConvertToInt64(TLink value) =>
1173         ↪ _addressToInt64Converter.Convert(value);
1174
1175     /// <summary>
1176     /// <para>
1177     /// Converts the to address using the specified value.
1178     /// </para>
1179     /// <para></para>
1180     /// </summary>
1181     /// <param name="value">
1182     /// <para>The value.</para>
1183     /// <para></para>
1184     /// </param>
1185     /// <returns>
1186     /// <para>The link</para>
1187     /// <para></para>
1188     /// </returns>
1189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1190     protected virtual TLink ConvertToAddress(long value) =>
1191         ↪ _int64ToAddressConverter.Convert(value);
1192
1193     /// <summary>
1194     /// <para>
1195     /// Adds the first.
1196     /// </para>

```



```

1193     /// <para></para>
1194     /// </summary>
1195     /// <param name="first">
1196     /// <para>The first.</para>
1197     /// <para></para>
1198     /// </param>
1199     /// <param name="second">
1200     /// <para>The second.</para>
1201     /// <para></para>
1202     /// </param>
1203     /// <returns>
1204     /// <para>The link</para>
1205     /// <para></para>
1206     /// </returns>
1207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1208     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
1209         ↪ second);
1210
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 TLink Subtract(TLink first, TLink second) =>
1231         ↪ Arithmetic<TLink>.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 TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
1249
1250     /// <summary>
1251     /// <para>
1252     /// Decrements the link.
1253     /// </para>
1254     /// <para></para>
1255     /// </summary>
1256     /// <param name="link">
1257     /// <para>The link.</para>
1258     /// <para></para>
1259     /// </param>
1260     /// <returns>
1261     /// <para>The link</para>
1262     /// <para></para>
1263     /// </returns>
1264     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1265     protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
1266
1267     #region Disposable
1268     /// <summary>
1269     /// <para>

```

```

1269     /// Gets the allow multiple dispose calls value.
1270     /// </para>
1271     /// <para></para>
1272     /// </summary>
1273     protected override bool AllowMultipleDisposeCalls
1274     {
1275         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1276         get => true;
1277     }
1278
1279     /// <summary>
1280     /// <para>
1281     /// Disposes the manual.
1282     /// </para>
1283     /// <para></para>
1284     /// </summary>
1285     /// <param name="manual">
1286     /// <para>The manual.</para>
1287     /// <para></para>
1288     /// </param>
1289     /// <param name="wasDisposed">
1290     /// <para>The was disposed.</para>
1291     /// <para></para>
1292     /// </param>
1293     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1294     protected override void Dispose(bool manual, bool wasDisposed)
1295     {
1296         if (!wasDisposed)
1297         {
1298             ResetPointers();
1299             _dataMemory.DisposeIfPossible();
1300             _indexMemory.DisposeIfPossible();
1301         }
1302     }
1303
1304     #endregion
1305 }
1306 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Methods.Lists;
3  using Platform.Converters;
4  using static System.Runtime.CompilerServices.Unsafe;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split.Generic
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the unused links list methods.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="AbsoluteCircularDoublyLinkedListMethods{TLink}" />
17     /// <seealso cref="ILinksListMethods{TLink}" />
18     public unsafe class UnusedLinksListMethods<TLink> :
19     ↪ AbsoluteCircularDoublyLinkedListMethods<TLink>, ILinksListMethods<TLink>
20     {
21         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
22         ↪ UncheckedConverter<TLink, long>.Default;
23         private readonly byte* _links;
24         private readonly byte* _header;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UnusedLinksListMethods" /> instance.
29         /// </para>
30         /// <para></para>
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>

```

```

38 [MethodImpl(MethodImplOptions.AggressiveInlining)]
39 public UnusedLinksListMethods(byte* links, byte* header)
40 {
41     _links = links;
42     _header = header;
43 }
44
45 /// <summary>
46 /// <para>
47 /// Gets the header reference.
48 /// </para>
49 /// <para></para>
50 /// </summary>
51 /// <returns>
52 /// <para>A ref links header of t link</para>
53 /// <para></para>
54 /// </returns>
55 [MethodImpl(MethodImplOptions.AggressiveInlining)]
56 protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLink>>(_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<TLink> GetLinkDataPartReference(TLink link) => ref
    ↪ AsRef<RawLinkDataPart<TLink>>(_links + (RawLinkDataPart<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));
74
75 /// <summary>
76 /// <para>
77 /// Gets the first.
78 /// </para>
79 /// <para></para>
80 /// </summary>
81 /// <returns>
82 /// <para>The link</para>
83 /// <para></para>
84 /// </returns>
85 [MethodImpl(MethodImplOptions.AggressiveInlining)]
86 protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
87
88 /// <summary>
89 /// <para>
90 /// Gets the last.
91 /// </para>
92 /// <para></para>
93 /// </summary>
94 /// <returns>
95 /// <para>The link</para>
96 /// <para></para>
97 /// </returns>
98 [MethodImpl(MethodImplOptions.AggressiveInlining)]
99 protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
100
101 /// <summary>
102 /// <para>
103 /// Gets the previous using the specified element.
104 /// </para>
105 /// <para></para>
106 /// </summary>
107 /// <param name="element">
108 /// <para>The element.</para>
109 /// <para></para>
110 /// </param>
111 /// <returns>
112 /// <para>The link</para>

```

```

113     /// <para></para>
114     /// </returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     protected override TLink GetPrevious(TLink element) =>
117         ↪ GetLinkDataPartReference(element).Source;
118
119     /// <summary>
120     /// <para>
121     /// Gets the next using the specified element.
122     /// </para>
123     /// </summary>
124     /// <param name="element">
125     /// <para>The element.</para>
126     /// </param>
127     /// </returns>
128     /// <para>The link</para>
129     /// </returns>
130     [MethodImpl(MethodImplOptions.AggressiveInlining)]
131     protected override TLink GetNext(TLink element) =>
132         ↪ GetLinkDataPartReference(element).Target;
133
134     /// <summary>
135     /// <para>
136     /// Gets the size.
137     /// </para>
138     /// </summary>
139     /// </returns>
140     /// <para>The link</para>
141     /// </returns>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     protected override TLink GetSize() => GetHeaderReference().FreeLinks;
144
145     /// <summary>
146     /// <para>
147     /// Sets the first using the specified element.
148     /// </para>
149     /// </summary>
150     /// <param name="element">
151     /// <para>The element.</para>
152     /// </param>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
155         ↪ element;
156
157     /// <summary>
158     /// <para>
159     /// Sets the last using the specified element.
160     /// </para>
161     /// </summary>
162     /// <param name="element">
163     /// <para>The element.</para>
164     /// </param>
165     [MethodImpl(MethodImplOptions.AggressiveInlining)]
166     protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
167         ↪ element;
168
169     /// <summary>
170     /// <para>
171     /// Sets the previous using the specified element.
172     /// </para>
173     /// </summary>
174     /// <param name="element">
175     /// <para>The element.</para>
176     /// </param>
177     /// <param name="previous">
178     /// <para>The previous.</para>
179     /// </param>
180

```

```

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

```

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

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.Split
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link data part.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLinkDataPart<TLink> : IEquatable<RawLinkDataPart<TLink>>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
19             ↪ EqualityComparer<TLink>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLinkDataPart<TLink>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLink Source;
36
37         /// <summary>
38         /// <para>
39         /// The target.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink Target;
44     }

```

```

40     /// </summary>
41     public TLink Target;
42
43     /// <summary>
44     /// <para>
45     /// Determines whether this instance equals.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="obj">
50     /// <para>The obj.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>The bool</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public override bool Equals(object obj) => obj is RawLinkDataPart<TLink> link ?
        ↳ 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<TLink> 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<TLink> left, RawLinkDataPart<TLink>
        ↳ right) => left.Equals(right);
94
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public static bool operator !=(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
        ↳ 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<TLink> : IEquatable<RawLinkIndexPart<TLink>>
17  {
18      private static readonly EqualityComparer<TLink> _equalityComparer =
19          ↳ EqualityComparer<TLink>.Default;
20
21      /// <summary>
22      /// <para>
23      /// The size.
24      /// </para>
25      /// </summary>
26      public static readonly long SizeInBytes = Structure<RawLinkIndexPart<TLink>>.Size;
27
28      /// <summary>
29      /// <para>
30      /// The root as source.
31      /// </para>
32      /// <para></para>
33      /// </summary>
34      public TLink RootAsSource;
35      /// <summary>
36      /// <para>
37      /// The left as source.
38      /// </para>
39      /// <para></para>
40      /// </summary>
41      public TLink LeftAsSource;
42      /// <summary>
43      /// <para>
44      /// The right as source.
45      /// </para>
46      /// <para></para>
47      /// </summary>
48      public TLink RightAsSource;
49      /// <summary>
50      /// <para>
51      /// The size as source.
52      /// </para>
53      /// <para></para>
54      /// </summary>
55      public TLink SizeAsSource;
56      /// <summary>
57      /// <para>
58      /// The root as target.
59      /// </para>
60      /// <para></para>
61      /// </summary>
62      public TLink RootAsTarget;
63      /// <summary>
64      /// <para>
65      /// The left as target.
66      /// </para>
67      /// <para></para>
68      /// </summary>
69      public TLink LeftAsTarget;
70      /// <summary>
71      /// <para>
72      /// The right as target.
73      /// </para>
74      /// <para></para>
75      /// </summary>
76      public TLink RightAsTarget;
77      /// <summary>
78      /// <para>
79      /// The size as target.
80      /// </para>
81      /// <para></para>
82      /// </summary>
83      public TLink SizeAsTarget;
84
85      /// <summary>
86      /// <para>
87      /// Determines whether this instance equals.
88      /// </para>
89      /// <para></para>
90      /// </summary>
91      /// <param name="obj">
92      /// <para>The obj.</para>

```

```

93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public override bool Equals(object obj) => obj is RawLinkIndexPart<TLink> link ?
        ↳ Equals(link) : false;

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLink = System.UInt32;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 32 external links recursionless size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}">
16    /// <seealso cref="ILinksTreeMethods{TLink}">
17    public unsafe abstract class UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
        ↳ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>

```



```

18 {
19     /// <summary>
20     /// <para>
21     /// The links data parts.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26     /// <summary>
27     /// <para>
28     /// The links index parts.
29     /// </para>
30     /// <para></para>
31     /// </summary>
32     protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33     /// <summary>
34     /// <para>
35     /// The header.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected new readonly LinksHeader<TLink>* Header;
40
41     /// <summary>
42     /// <para>
43     /// Initializes a new <see
44     ↪ cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="constants">
49     /// <para>A constants.</para>
50     /// <para></para>
51     /// </param>
52     /// <param name="linksDataParts">
53     /// <para>A links data parts.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="linksIndexParts">
57     /// <para>A links index parts.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="header">
61     /// <para>A header.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected
66     ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
67     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
68     ↪ linksIndexParts, LinksHeader<TLink>* header)
69     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
70     {
71         LinksDataParts = linksDataParts;
72         LinksIndexParts = linksIndexParts;
73         Header = header;
74     }
75
76     /// <summary>
77     /// <para>
78     /// Gets the zero.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <returns>
83     /// <para>The link</para>
84     /// <para></para>
85     /// </returns>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     protected override TLink GetZero() => 0U;
88
89     /// <summary>
90     /// <para>
91     /// Determines whether this instance equal to zero.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="value">

```

```

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

```

```

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

```

```

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

```

```

323     /// <returns>
324     /// <para>The link</para>
325     /// <para></para>
326     /// </returns>
327     [MethodImpl(MethodImplOptions.AggressiveInlining)]
328     protected override TLink Add(TLink first, TLink second) => first + second;
329
330     /// <summary>
331     /// <para>
332     /// Subtracts the first.
333     /// </para>
334     /// <para></para>
335     /// </summary>
336     /// <param name="first">
337     /// <para>The first.</para>
338     /// <para></para>
339     /// </param>
340     /// <param name="second">
341     /// <para>The second.</para>
342     /// <para></para>
343     /// </param>
344     /// <returns>
345     /// <para>The link</para>
346     /// <para></para>
347     /// </returns>
348     [MethodImpl(MethodImplOptions.AggressiveInlining)]
349     protected override TLink Subtract(TLink first, TLink second) => first - second;
350
351     /// <summary>
352     /// <para>
353     /// Gets the header reference.
354     /// </para>
355     /// <para></para>
356     /// </summary>
357     /// <returns>
358     /// <para>A ref links header of t link</para>
359     /// <para></para>
360     /// </returns>
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;
363
364     /// <summary>
365     /// <para>
366     /// Gets the link data part reference using the specified link.
367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="link">
371     /// <para>The link.</para>
372     /// <para></para>
373     /// </param>
374     /// <returns>
375     /// <para>A ref raw link data part of t link</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
380         ↪ ref LinksDataParts[link];
381
382     /// <summary>
383     /// <para>
384     /// Gets the link index part reference using the specified link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>A ref raw link index part of t link</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
398         ↪ ref LinksIndexParts[link];
399
400     /// <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(TLink first, TLink second)
418     {
419         ref var firstLink = ref LinksDataParts[first];
420         ref var secondLink = ref LinksDataParts[second];
421         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
422             ↪ secondLink.Source, secondLink.Target);
423     }
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance first is to the right of second.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The bool</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
444     {
445         ref var firstLink = ref LinksDataParts[first];
446         ref var secondLink = ref LinksDataParts[second];
447         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
448             ↪ secondLink.Source, secondLink.Target);
449     }
450 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLink = System.UInt32;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 32 external links size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16    /// <seealso cref="ILinksTreeMethods{TLink}"/>
17    public unsafe abstract class UInt32ExternalLinksSizeBalancedTreeMethodsBase :
18        ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
19    {
20        /// <summary>
21        /// <para>
22        /// The links data parts.

```

```

22     /// </para>
23     /// <para></para>
24     /// </summary>
25     protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26     /// <summary>
27     /// <para>
28     /// The links index parts.
29     /// </para>
30     /// <para></para>
31     /// </summary>
32     protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33     /// <summary>
34     /// <para>
35     /// The header.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected new readonly LinksHeader<TLink>* Header;
40
41     /// <summary>
42     /// <para>
43     /// Initializes a new <see cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
44     ↪ instance.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="constants">
49     /// <para>A constants.</para>
50     /// <para></para>
51     /// </param>
52     /// <param name="linksDataParts">
53     /// <para>A links data parts.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="linksIndexParts">
57     /// <para>A links index parts.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="header">
61     /// <para>A header.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected UInt32ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
66     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
67     ↪ linksIndexParts, LinksHeader<TLink>* header)
68     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
69     {
70         LinksDataParts = linksDataParts;
71         LinksIndexParts = linksIndexParts;
72         Header = header;
73     }
74
75     /// <summary>
76     /// <para>
77     /// Gets the zero.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLink GetZero() => 0U;
87
88     /// <summary>
89     /// <para>
90     /// Determines whether this instance equal to zero.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="value">
95     /// <para>The value.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The bool</para>

```

```

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

```



```

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

```

```

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

```

```

328     protected override TLink Add(TLink first, TLink second) => first + second;
329
330     /// <summary>
331     /// <para>
332     /// Subtracts the first.
333     /// </para>
334     /// <para></para>
335     /// </summary>
336     /// <param name="first">
337     /// <para>The first.</para>
338     /// <para></para>
339     /// </param>
340     /// <param name="second">
341     /// <para>The second.</para>
342     /// <para></para>
343     /// </param>
344     /// <returns>
345     /// <para>The link</para>
346     /// <para></para>
347     /// </returns>
348     [MethodImpl(MethodImplOptions.AggressiveInlining)]
349     protected override TLink Subtract(TLink first, TLink second) => first - second;
350
351     /// <summary>
352     /// <para>
353     /// Gets the header reference.
354     /// </para>
355     /// <para></para>
356     /// </summary>
357     /// <returns>
358     /// <para>A ref links header of t link</para>
359     /// <para></para>
360     /// </returns>
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;
363
364     /// <summary>
365     /// <para>
366     /// Gets the link data part reference using the specified link.
367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="link">
371     /// <para>The link.</para>
372     /// <para></para>
373     /// </param>
374     /// <returns>
375     /// <para>A ref raw link data part of t link</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
380         ↪ ref LinksDataParts[link];
381
382     /// <summary>
383     /// <para>
384     /// Gets the link index part reference using the specified link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>A ref raw link index part of t link</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
398         ↪ ref LinksIndexParts[link];
399
400     /// <summary>
401     /// <para>
402     /// Determines whether this instance first is to the left of second.
403     /// </para>
404     /// <para></para>
405     /// </summary>

```

```

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

```

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

```

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

```

```

26     /// </param>
27     /// <param name="linksDataParts">
28     /// <para>A links data parts.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public
41     ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
42     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
43     ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
44     ↪ linksIndexParts, header) { }
45
46     /// <summary>
47     /// <para>
48     /// Gets the left reference using the specified node.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="node">
53     /// <para>The node.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>The ref link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref TLink GetLeftReference(TLink node) => ref
62     ↪ LinksIndexParts[node].LeftAsSource;
63
64     /// <summary>
65     /// <para>
66     /// Gets the right reference using the specified node.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="node">
71     /// <para>The node.</para>
72     /// <para></para>
73     /// </param>
74     /// <returns>
75     /// <para>The ref link</para>
76     /// <para></para>
77     /// </returns>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected override ref TLink GetRightReference(TLink node) => ref
80     ↪ LinksIndexParts[node].RightAsSource;
81
82     /// <summary>
83     /// <para>
84     /// Gets the left using the specified node.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="node">
89     /// <para>The node.</para>
90     /// <para></para>
91     /// </param>
92     /// <returns>
93     /// <para>The link</para>
94     /// <para></para>
95     /// </returns>
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
98
99     /// <summary>
100    /// <para>
101    /// Gets the right using the specified node.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    /// <para></para>

```

```

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

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links sources size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksSourcesSizeBalancedTreeMethods :
16         ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32ExternalLinksSourcesSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
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 UInt32ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
        ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
        ↳ linksIndexParts, header) { }
41
42     /// <summary>
43     /// <para>
44     /// Gets the left reference using the specified node.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="node">
49     /// <para>The node.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLink GetLeftReference(TLink node) => ref
        ↳ LinksIndexParts[node].LeftAsSource;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
        ↳ LinksIndexParts[node].RightAsSource;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>

```

```

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

```

```

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

```

```

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

```

#### 1.54 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10      /// Represents the int 32 external links targets recursionless size balanced tree methods.
11      /// </para>
12      /// <para></para>
13      /// </summary>
14      /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15      public unsafe class UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16         ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17      {
18          /// <summary>
19          /// <para>
20          /// Initializes a new <see
21          ↪ cref="UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22          /// </para>
23          /// <para></para>
24          /// </summary>
25          /// <param name="constants">
26          /// <para>A constants.</para>
27          /// <para></para>
28          /// </param>
29          /// <param name="linksDataParts">
30          /// <para>A links data parts.</para>
31          /// <para></para>
32          /// </param>
33          /// <param name="linksIndexParts">
34          /// <para>A links index parts.</para>
35          /// <para></para>
36          /// </param>
37          /// <param name="header">
38          /// <para>A header.</para>
39          /// <para></para>
40          /// </param>
41          [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

40 public
    ↳ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↳ linksIndexParts, header) { }
41
42 /// <summary>
43 /// <para>
44 /// Gets the left reference using the specified node.
45 /// </para>
46 /// <para></para>
47 /// </summary>
48 /// <param name="node">
49 /// <para>The node.</para>
50 /// <para></para>
51 /// </param>
52 /// <returns>
53 /// <para>The ref link</para>
54 /// <para></para>
55 /// </returns>
56 [MethodImpl(MethodImplOptions.AggressiveInlining)]
57 protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ LinksIndexParts[node].LeftAsTarget;
58
59 /// <summary>
60 /// <para>
61 /// Gets the right reference using the specified node.
62 /// </para>
63 /// <para></para>
64 /// </summary>
65 /// <param name="node">
66 /// <para>The node.</para>
67 /// <para></para>
68 /// </param>
69 /// <returns>
70 /// <para>The ref link</para>
71 /// <para></para>
72 /// </returns>
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 protected override ref TLink GetRightReference(TLink node) => ref
    ↳ LinksIndexParts[node].RightAsTarget;
75
76 /// <summary>
77 /// <para>
78 /// Gets the left using the specified node.
79 /// </para>
80 /// <para></para>
81 /// </summary>
82 /// <param name="node">
83 /// <para>The node.</para>
84 /// <para></para>
85 /// </param>
86 /// <returns>
87 /// <para>The link</para>
88 /// <para></para>
89 /// </returns>
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93 /// <summary>
94 /// <para>
95 /// Gets the right using the specified node.
96 /// </para>
97 /// <para></para>
98 /// </summary>
99 /// <param name="node">
100 /// <para>The node.</para>
101 /// <para></para>
102 /// </param>
103 /// <returns>
104 /// <para>The link</para>
105 /// <para></para>
106 /// </returns>
107 [MethodImpl(MethodImplOptions.AggressiveInlining)]
108 protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110 /// <summary>
111 /// <para>

```

```

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

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksTargetsSizeBalancedTreeMethods :
16         ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32ExternalLinksTargetsSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt32ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
43             ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44             ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45             ↪ linksIndexParts, header) { }

```



```

46     /// <para></para>
47     /// </summary>
48     /// <param name="node">
49     /// <para>The node.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLink GetLeftReference(TLink node) => ref
    ↪ LinksIndexParts[node].LeftAsTarget;

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

75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>

```

```

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

```

```

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

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLink = System.UInt32;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 32 internal links recursionless size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
16    public unsafe abstract class UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
18    {
19        /// <summary>
20        /// <para>
21        /// The links data parts.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26        /// <summary>
27        /// <para>
28        /// The links index parts.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33        /// <summary>
34        /// <para>
35        /// The header.
36        /// </para>
37        /// <para></para>
38        /// </summary>
39        protected new readonly LinksHeader<TLink>* Header;
40        /// <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        /// <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">

```

```

59     /// <para>A header.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected
64     ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
65     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
66     ↪ linksIndexParts, LinksHeader<TLink>* header)
67     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
68     {
69         LinksDataParts = linksDataParts;
70         LinksIndexParts = linksIndexParts;
71         Header = header;
72     }
73
74     /// <summary>
75     /// <para>
76     /// Gets the zero.
77     /// </para>
78     /// </summary>
79     /// <returns>
80     /// <para>The link</para>
81     /// <para></para>
82     /// </returns>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     protected override TLink GetZero() => 0U;
85
86     /// <summary>
87     /// <para>
88     /// Determines whether this instance equal to zero.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="value">
93     /// <para>The value.</para>
94     /// <para></para>
95     /// </param>
96     /// <returns>
97     /// <para>The bool</para>
98     /// <para></para>
99     /// </returns>
100     [MethodImpl(MethodImplOptions.AggressiveInlining)]
101     protected override bool EqualToZero(TLink value) => value == 0U;
102
103     /// <summary>
104     /// <para>
105     /// Determines whether this instance are equal.
106     /// </para>
107     /// <para></para>
108     /// </summary>
109     /// <param name="first">
110     /// <para>The first.</para>
111     /// <para></para>
112     /// </param>
113     /// <param name="second">
114     /// <para>The second.</para>
115     /// <para></para>
116     /// </param>
117     /// <returns>
118     /// <para>The bool</para>
119     /// <para></para>
120     /// </returns>
121     [MethodImpl(MethodImplOptions.AggressiveInlining)]
122     protected override bool AreEqual(TLink first, TLink second) => first == second;
123
124     /// <summary>
125     /// <para>
126     /// Determines whether this instance greater than zero.
127     /// </para>
128     /// <para></para>
129     /// </summary>
130     /// <param name="value">
131     /// <para>The value.</para>
132     /// <para></para>
133     /// </param>
134     /// <returns>
135     /// <para>The bool</para>

```

```

134    /// <para></para>
135    /// </returns>
136    [MethodImpl(MethodImplOptions.AggressiveInlining)]
137    protected override bool GreaterThanZero(TLink value) => value > 0U;
138
139    /// <summary>
140    /// <para>
141    /// Determines whether this instance greater than.
142    /// </para>
143    /// <para></para>
144    /// </summary>
145    /// <param name="first">
146    /// <para>The first.</para>
147    /// <para></para>
148    /// </param>
149    /// <param name="second">
150    /// <para>The second.</para>
151    /// <para></para>
152    /// </param>
153    /// <returns>
154    /// <para>The bool</para>
155    /// <para></para>
156    /// </returns>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override bool GreaterThan(TLink first, TLink second) => first > second;
159
160    /// <summary>
161    /// <para>
162    /// Determines whether this instance greater or equal than.
163    /// </para>
164    /// <para></para>
165    /// </summary>
166    /// <param name="first">
167    /// <para>The first.</para>
168    /// <para></para>
169    /// </param>
170    /// <param name="second">
171    /// <para>The second.</para>
172    /// <para></para>
173    /// </param>
174    /// <returns>
175    /// <para>The bool</para>
176    /// <para></para>
177    /// </returns>
178    [MethodImpl(MethodImplOptions.AggressiveInlining)]
179    protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
180
181    /// <summary>
182    /// <para>
183    /// Determines whether this instance greater or equal than zero.
184    /// </para>
185    /// <para></para>
186    /// </summary>
187    /// <param name="value">
188    /// <para>The value.</para>
189    /// <para></para>
190    /// </param>
191    /// <returns>
192    /// <para>The bool</para>
193    /// <para></para>
194    /// </returns>
195    [MethodImpl(MethodImplOptions.AggressiveInlining)]
196    protected override bool GreaterOrEqualThanZero(TLink value) => true; // value >= 0 is
    ↪ 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(TLink value) => value == 0UL; // value is
    ↪ always >= 0 for ulong
214
215    /// <summary>
216    /// <para>
217    /// Determines whether this instance less or equal than.
218    /// </para>
219    /// <para></para>
220    /// </summary>
221    /// <param name="first">
222    /// <para>The first.</para>
223    /// <para></para>
224    /// </param>
225    /// <param name="second">
226    /// <para>The second.</para>
227    /// <para></para>
228    /// </param>
229    /// <returns>
230    /// <para>The bool</para>
231    /// <para></para>
232    /// </returns>
233    [MethodImpl(MethodImplOptions.AggressiveInlining)]
234    protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;
235
236    /// <summary>
237    /// <para>
238    /// Determines whether this instance less than zero.
239    /// </para>
240    /// <para></para>
241    /// </summary>
242    /// <param name="value">
243    /// <para>The value.</para>
244    /// <para></para>
245    /// </param>
246    /// <returns>
247    /// <para>The bool</para>
248    /// <para></para>
249    /// </returns>
250    [MethodImpl(MethodImplOptions.AggressiveInlining)]
251    protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
    ↪ for ulong
252
253    /// <summary>
254    /// <para>
255    /// Determines whether this instance less than.
256    /// </para>
257    /// <para></para>
258    /// </summary>
259    /// <param name="first">
260    /// <para>The first.</para>
261    /// <para></para>
262    /// </param>
263    /// <param name="second">
264    /// <para>The second.</para>
265    /// <para></para>
266    /// </param>
267    /// <returns>
268    /// <para>The bool</para>
269    /// <para></para>
270    /// </returns>
271    [MethodImpl(MethodImplOptions.AggressiveInlining)]
272    protected override bool LessThan(TLink first, TLink second) => first < second;
273
274    /// <summary>
275    /// <para>
276    /// Increments the value.
277    /// </para>
278    /// <para></para>
279    /// </summary>
280    /// <param name="value">
281    /// <para>The value.</para>
282    /// <para></para>
283    /// </param>
284    /// <returns>
285    /// <para>The link</para>
286    /// <para></para>

```

```

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

```



```

365     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
366         ↪ ref LinksDataParts[link];
367
368     /// <summary>
369     /// <para>
370     /// Gets the link index part reference using the specified link.
371     /// </para>
372     /// </summary>
373     /// <param name="link">
374     /// <para>The link.</para>
375     /// </para>
376     /// </param>
377     /// <returns>
378     /// <para>A ref raw link index part of t link</para>
379     /// </returns>
380     [MethodImpl(MethodImplOptions.AggressiveInlining)]
381     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
382         ↪ ref LinksIndexParts[link];
383
384     /// <summary>
385     /// <para>
386     /// Determines whether this instance first is to the left of second.
387     /// </para>
388     /// </summary>
389     /// <param name="first">
390     /// <para>The first.</para>
391     /// </para>
392     /// </param>
393     /// <param name="second">
394     /// <para>The second.</para>
395     /// </para>
396     /// </param>
397     /// <returns>
398     /// <para>The bool</para>
399     /// </returns>
400     [MethodImpl(MethodImplOptions.AggressiveInlining)]
401     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
402         ↪ GetKeyPartValue(first) < GetKeyPartValue(second);
403
404     /// <summary>
405     /// <para>
406     /// Determines whether this instance first is to the right of second.
407     /// </para>
408     /// </summary>
409     /// <param name="first">
410     /// <para>The first.</para>
411     /// </para>
412     /// </param>
413     /// <param name="second">
414     /// <para>The second.</para>
415     /// </para>
416     /// </param>
417     /// <returns>
418     /// <para>The bool</para>
419     /// </returns>
420     [MethodImpl(MethodImplOptions.AggressiveInlining)]
421     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
422         ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
423
424     }
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 TLink = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>

```

```

11  /// Represents the int 32 internal links size balanced tree methods base.
12  /// </para>
13  /// <para></para>
14  /// </summary>
15  /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16  public unsafe abstract class UInt32InternalLinksSizeBalancedTreeMethodsBase :
    ↳ InternalLinksSizeBalancedTreeMethodsBase<TLink>
17  {
18      /// <summary>
19      /// <para>
20      /// The links data parts.
21      /// </para>
22      /// <para></para>
23      /// </summary>
24      protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
25      /// <summary>
26      /// <para>
27      /// The links index parts.
28      /// </para>
29      /// <para></para>
30      /// </summary>
31      protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
32      /// <summary>
33      /// <para>
34      /// The header.
35      /// </para>
36      /// <para></para>
37      /// </summary>
38      protected new readonly LinksHeader<TLink>* Header;
39
40      /// <summary>
41      /// <para>
42      /// Initializes a new <see cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
43      ↳ instance.
44      /// </para>
45      /// <para></para>
46      /// </summary>
47      /// <param name="constants">
48      /// <para>A constants.</para>
49      /// <para></para>
50      /// </param>
51      /// <param name="linksDataParts">
52      /// <para>A links data parts.</para>
53      /// <para></para>
54      /// </param>
55      /// <param name="linksIndexParts">
56      /// <para>A links index parts.</para>
57      /// <para></para>
58      /// </param>
59      /// <param name="header">
60      /// <para>A header.</para>
61      /// <para></para>
62      /// </param>
63      [MethodImpl(MethodImplOptions.AggressiveInlining)]
64      protected UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↳ linksIndexParts, LinksHeader<TLink>* header)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65      {
66          LinksDataParts = linksDataParts;
67          LinksIndexParts = linksIndexParts;
68          Header = header;
69      }
70
71      /// <summary>
72      /// <para>
73      /// Gets the zero.
74      /// </para>
75      /// <para></para>
76      /// </summary>
77      /// <returns>
78      /// <para>The link</para>
79      /// <para></para>
80      /// </returns>
81      [MethodImpl(MethodImplOptions.AggressiveInlining)]
82      protected override TLink GetZero() => 0U;
83
84      /// <summary>

```

```

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

```

```

163     /// </para>
164     /// <para></para>
165     /// </summary>
166     /// <param name="first">
167     /// <para>The first.</para>
168     /// <para></para>
169     /// </param>
170     /// <param name="second">
171     /// <para>The second.</para>
172     /// <para></para>
173     /// </param>
174     /// <returns>
175     /// <para>The bool</para>
176     /// <para></para>
177     /// </returns>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
179     protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
180
181     /// <summary>
182     /// <para>
183     /// Determines whether this instance greater or equal than zero.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="value">
188     /// <para>The value.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The bool</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override bool GreaterOrEqualThanZero(TLink value) => true; // value >= 0 is
    ↪ 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(TLink 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(TLink first, TLink second) => first <= second;
235
236     /// <summary>
237     /// <para>
238     /// Determines whether this instance less than zero.

```

```

239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="value">
243     /// <para>The value.</para>
244     /// <para></para>
245     /// </param>
246     /// <returns>
247     /// <para>The bool</para>
248     /// <para></para>
249     /// </returns>
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
    ↪    for ulong

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

```

```

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

```

```

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

```

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

```

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

```

```

39     public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
40         ↳ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
41         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
42     {
43         _linksDataParts = linksDataParts;
44         _linksIndexParts = linksIndexParts;
45     }
46
47     /// <summary>
48     /// <para>
49     /// Gets the link data part reference using the specified link.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     /// <param name="link">
54     /// <para>The link.</para>
55     /// <para></para>
56     /// </param>
57     /// <returns>
58     /// <para>A ref raw link data part of t link</para>
59     /// <para></para>
60     /// </returns>
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
63         ↳ ref _linksDataParts[link];
64
65     /// <summary>
66     /// <para>
67     /// Gets the link index part reference using the specified link.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <param name="link">
72     /// <para>The link.</para>
73     /// <para></para>
74     /// </param>
75     /// <returns>
76     /// <para>A ref raw link index part of t link</para>
77     /// <para></para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
81         ↳ ref _linksIndexParts[link];
82 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
16         ↳ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↳ cref="UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>

```



```

30    /// </param>
31    /// <param name="linksIndexParts">
32    /// <para>A links index parts.</para>
33    /// <para></para>
34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public
    ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
    ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↪ linksIndexParts, header) { }
41
42    /// <summary>
43    /// <para>
44    /// Gets the left reference using the specified node.
45    /// </para>
46    /// <para></para>
47    /// </summary>
48    /// <param name="node">
49    /// <para>The node.</para>
50    /// <para></para>
51    /// </param>
52    /// <returns>
53    /// <para>The ref link</para>
54    /// <para></para>
55    /// </returns>
56    [MethodImpl(MethodImplOptions.AggressiveInlining)]
57    protected override ref TLink GetLeftReference(TLink node) => ref
    ↪ LinksIndexParts[node].LeftAsSource;
58
59    /// <summary>
60    /// <para>
61    /// Gets the right reference using the specified node.
62    /// </para>
63    /// <para></para>
64    /// </summary>
65    /// <param name="node">
66    /// <para>The node.</para>
67    /// <para></para>
68    /// </param>
69    /// <returns>
70    /// <para>The ref link</para>
71    /// <para></para>
72    /// </returns>
73    [MethodImpl(MethodImplOptions.AggressiveInlining)]
74    protected override ref TLink GetRightReference(TLink node) => ref
    ↪ LinksIndexParts[node].RightAsSource;
75
76    /// <summary>
77    /// <para>
78    /// Gets the left using the specified node.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <param name="node">
83    /// <para>The node.</para>
84    /// <para></para>
85    /// </param>
86    /// <returns>
87    /// <para>The link</para>
88    /// <para></para>
89    /// </returns>
90    [MethodImpl(MethodImplOptions.AggressiveInlining)]
91    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>

```

```

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

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links sources size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksSourcesSizeBalancedTreeMethods :
        ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32InternalLinksSourcesSizeBalancedTreeMethods"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UInt32InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
        ↪ linksIndexParts, header) { }
42
43         /// <summary>
44         /// <para>
45         /// Gets the left reference using the specified node.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <param name="node">
50         /// <para>The node.</para>
51         /// <para></para>
52         /// </param>
53         /// <returns>
54         /// <para>The ref link</para>
55         /// <para></para>
56         /// </returns>
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

57     protected override ref TLink GetLeftReference(TLink 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 TLink GetRightReference(TLink 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 TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
94
95     /// <summary>
96     /// <para>
97     /// Gets the right using the specified node.
98     /// </para>
99     /// <para></para>
100    /// </summary>
101    /// <param name="node">
102    /// <para>The node.</para>
103    /// <para></para>
104    /// </param>
105    /// <returns>
106    /// <para>The link</para>
107    /// <para></para>
108    /// </returns>
109    [MethodImpl(MethodImplOptions.AggressiveInlining)]
110    protected override TLink GetRight(TLink node) => 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(TLink node, TLink left) =>
128        ↳ LinksIndexParts[node].LeftAsSource = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>

```

```

132     /// </summary>
133     /// <param name="node">
134     /// <para>The node.</para>
135     /// <para></para>
136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLink node, TLink right) =>
143         ↳ LinksIndexParts[node].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 TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
161
162     /// <summary>
163     /// <para>
164     /// Sets the size using the specified node.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="node">
169     /// <para>The node.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="size">
173     /// <para>The size.</para>
174     /// <para></para>
175     /// </param>
176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
177     protected override void SetSize(TLink node, TLink size) =>
178         ↳ LinksIndexParts[node].SizeAsSource = size;
179
180     /// <summary>
181     /// <para>
182     /// Gets the tree root using the specified node.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <param name="node">
187     /// <para>The node.</para>
188     /// <para></para>
189     /// </param>
190     /// <returns>
191     /// <para>The link</para>
192     /// <para></para>
193     /// </returns>
194     [MethodImpl(MethodImplOptions.AggressiveInlining)]
195     protected override TLink GetTreeRoot(TLink node) => 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>

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>

```

```

15 public unsafe class UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↳ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16 {
17     /// <summary>
18     /// <para>
19     /// Initializes a new <see
    ↳ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20     /// </para>
21     /// <para></para>
22     /// </summary>
23     /// <param name="constants">
24     /// <para>A constants.</para>
25     /// <para></para>
26     /// </param>
27     /// <param name="linksDataParts">
28     /// <para>A links data parts.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public
    ↳ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↳ linksIndexParts, header) { }
41
42     /// <summary>
43     /// <para>
44     /// Gets the left reference using the specified node.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="node">
49     /// <para>The node.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ LinksIndexParts[node].LeftAsTarget;
58
59     /// <summary>
60     /// <para>
61     /// Gets the right reference using the specified node.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
    ↳ LinksIndexParts[node].RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>

```



```

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

```

```

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

```

```

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

```

## 1.62 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethod

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksTargetsSizeBalancedTreeMethods :
16         ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32InternalLinksTargetsSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt32InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
43             ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44             ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45             ↪ linksIndexParts, header) { }

```

```

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

```

```

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

```

```

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

```

```

267 }
268 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLink = System.UInt32;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the int 32 split memory links.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="SplitMemoryLinksBase{TLink}" />
19     public unsafe class UInt32SplitMemoryLinks : SplitMemoryLinksBase<TLink>
20     {
21         private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
24         private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
25         private LinksHeader<TLink>* _header;
26         private RawLinkDataPart<TLink>* _linksDataParts;
27         private RawLinkIndexPart<TLink>* _linksIndexParts;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="UInt32SplitMemoryLinks" /> instance.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         /// <param name="dataMemory">
36         /// <para>A data memory.</para>
37         /// <para></para>
38         /// </param>
39         /// <param name="indexMemory">
40         /// <para>A index memory.</para>
41         /// <para></para>
42         /// </param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
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<TLink>>.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>

```

```

75     /// <para></para>
76     /// </param>
77     /// <param name="indexMemory">
78     /// <para>A index memory.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="memoryReservationStep">
82     /// <para>A memory reservation step.</para>
83     /// <para></para>
84     /// </param>
85     /// <param name="constants">
86     /// <para>A constants.</para>
87     /// <para></para>
88     /// </param>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
        ↪ this(dataMemory, indexMemory, memoryReservationStep, constants,
        ↪ IndexTreeType.Default, useLinkedList: true) { }

91
92     /// <summary>
93     /// <para>
94     /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <param name="dataMemory">
99     /// <para>A data memory.</para>
100    /// <para></para>
101    /// </param>
102    /// <param name="indexMemory">
103    /// <para>A index memory.</para>
104    /// <para></para>
105    /// </param>
106    /// <param name="memoryReservationStep">
107    /// <para>A memory reservation step.</para>
108    /// <para></para>
109    /// </param>
110    /// <param name="constants">
111    /// <para>A constants.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="indexTreeType">
115    /// <para>A index tree type.</para>
116    /// <para></para>
117    /// </param>
118    /// <param name="useLinkedList">
119    /// <para>A use linked list.</para>
120    /// <para></para>
121    /// </param>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
        ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↪ memoryReservationStep, constants, useLinkedList)
124    {
125        if (indexTreeType == IndexTreeType.SizeBalancedTree)
126        {
127            _createInternalSourceTreeMethods = () => new
                ↪ UInt32InternalLinksSourcesSizeBalancedTreeMethods(Constants,
                ↪ _linksDataParts, _linksIndexParts, _header);
128            _createExternalSourceTreeMethods = () => new
                ↪ UInt32ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
                ↪ _linksDataParts, _linksIndexParts, _header);
129            _createInternalTargetTreeMethods = () => new
                ↪ 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,
136             ↳ _linksDataParts, _linksIndexParts, _header);
        _createInternalTargetTreeMethods = () => new
            ↳ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
137         _createExternalTargetTreeMethods = () => new
            ↳ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
138     }
139     Init(dataMemory, indexMemory);
140 }
141
142 /// <summary>
143 /// <para>
144 /// Sets the pointers using the specified data memory.
145 /// </para>
146 /// <para></para>
147 /// </summary>
148 /// <param name="dataMemory">
149 /// <para>The data memory.</para>
150 /// <para></para>
151 /// </param>
152 /// <param name="indexMemory">
153 /// <para>The index memory.</para>
154 /// <para></para>
155 /// </param>
156 [MethodImpl(MethodImplOptions.AggressiveInlining)]
157 protected override void SetPointers(IResizableDirectMemory dataMemory,
    ↳ IResizableDirectMemory indexMemory)
158 {
159     _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
160     _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
161     _header = (LinksHeader<TLink>*)indexMemory.Pointer;
162     if (_useLinkedList)
163     {
164         InternalSourcesListMethods = new
            ↳ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
            ↳ _linksIndexParts);
165     }
166     else
167     {
168         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
169     }
170     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
171     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
172     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
173     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
174 }
175
176 /// <summary>
177 /// <para>
178 /// Resets the pointers.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 [MethodImpl(MethodImplOptions.AggressiveInlining)]
183 protected override void ResetPointers()
184 {
185     base.ResetPointers();
186     _linksDataParts = null;
187     _linksIndexParts = null;
188     _header = null;
189 }
190
191 /// <summary>
192 /// <para>
193 /// Gets the header reference.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <returns>
198 /// <para>A ref links header of t link</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
203

```

```

204    /// <summary>
205    /// <para>
206    /// Gets the link data part reference using the specified link index.
207    /// </para>
208    /// <para></para>
209    /// </summary>
210    /// <param name="linkIndex">
211    /// <para>The link index.</para>
212    /// <para></para>
213    /// </param>
214    /// <returns>
215    /// <para>A ref raw link data part of t link</para>
216    /// <para></para>
217    /// </returns>
218    [MethodImpl(MethodImplOptions.AggressiveInlining)]
219    protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
220    ↪ => ref _linksDataParts[linkIndex];
221
222    /// <summary>
223    /// <para>
224    /// Gets the link index part reference using the specified link index.
225    /// </para>
226    /// <para></para>
227    /// </summary>
228    /// <param name="linkIndex">
229    /// <para>The link index.</para>
230    /// <para></para>
231    /// </param>
232    /// <returns>
233    /// <para>A ref raw link index part of t link</para>
234    /// <para></para>
235    /// </returns>
236    [MethodImpl(MethodImplOptions.AggressiveInlining)]
237    protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
238    ↪ linkIndex) => ref _linksIndexParts[linkIndex];
239
240    /// <summary>
241    /// <para>
242    /// Determines whether this instance are equal.
243    /// </para>
244    /// <para></para>
245    /// </summary>
246    /// <param name="first">
247    /// <para>The first.</para>
248    /// <para></para>
249    /// </param>
250    /// <param name="second">
251    /// <para>The second.</para>
252    /// <para></para>
253    /// </param>
254    /// <returns>
255    /// <para>The bool</para>
256    /// <para></para>
257    /// </returns>
258    [MethodImpl(MethodImplOptions.AggressiveInlining)]
259    protected override bool AreEqual(TLink first, TLink second) => first == second;
260
261    /// <summary>
262    /// <para>
263    /// Determines whether this instance less than.
264    /// </para>
265    /// <para></para>
266    /// </summary>
267    /// <param name="first">
268    /// <para>The first.</para>
269    /// <para></para>
270    /// </param>
271    /// <param name="second">
272    /// <para>The second.</para>
273    /// <para></para>
274    /// </param>
275    /// <returns>
276    /// <para>The bool</para>
277    /// <para></para>
278    /// </returns>
279    [MethodImpl(MethodImplOptions.AggressiveInlining)]
280    protected override bool LessThan(TLink first, TLink second) => first < second;

```

```

280    /// <summary>
281    /// <para>
282    /// Determines whether this instance less or equal than.
283    /// </para>
284    /// <para></para>
285    /// </summary>
286    /// <param name="first">
287    /// <para>The first.</para>
288    /// <para></para>
289    /// </param>
290    /// <param name="second">
291    /// <para>The second.</para>
292    /// <para></para>
293    /// </param>
294    /// <returns>
295    /// <para>The bool</para>
296    /// <para></para>
297    /// </returns>
298    [MethodImpl(MethodImplOptions.AggressiveInlining)]
299    protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;
300
301    /// <summary>
302    /// <para>
303    /// Determines whether this instance greater than.
304    /// </para>
305    /// <para></para>
306    /// </summary>
307    /// <param name="first">
308    /// <para>The first.</para>
309    /// <para></para>
310    /// </param>
311    /// <param name="second">
312    /// <para>The second.</para>
313    /// <para></para>
314    /// </param>
315    /// <returns>
316    /// <para>The bool</para>
317    /// <para></para>
318    /// </returns>
319    [MethodImpl(MethodImplOptions.AggressiveInlining)]
320    protected override bool GreaterThan(TLink first, TLink second) => first > second;
321
322    /// <summary>
323    /// <para>
324    /// Determines whether this instance greater or equal than.
325    /// </para>
326    /// <para></para>
327    /// </summary>
328    /// <param name="first">
329    /// <para>The first.</para>
330    /// <para></para>
331    /// </param>
332    /// <param name="second">
333    /// <para>The second.</para>
334    /// <para></para>
335    /// </param>
336    /// <returns>
337    /// <para>The bool</para>
338    /// <para></para>
339    /// </returns>
340    [MethodImpl(MethodImplOptions.AggressiveInlining)]
341    protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
342
343    /// <summary>
344    /// <para>
345    /// Gets the zero.
346    /// </para>
347    /// <para></para>
348    /// </summary>
349    /// <returns>
350    /// <para>The link</para>
351    /// <para></para>
352    /// </returns>
353    [MethodImpl(MethodImplOptions.AggressiveInlining)]
354    protected override TLink GetZero() => OU;
355
356    /// <summary>
357    /// <para>

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt32;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 unused links list methods.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="UnusedLinksListMethods{TLink}" />
16     public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<TLink>
17     {
18         private readonly RawLinkDataPart<TLink>* _links;
19         private readonly LinksHeader<TLink>* _header;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt32UnusedLinksListMethods" /> instance.
24         /// </para>
25         /// <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<TLink>* links, LinksHeader<TLink>*
    ↪ header)
37         : base((byte*)links, (byte*)header)
38     {
39         _links = links;
40         _header = header;
41     }
42
43     /// <summary>
44     /// <para>
45     /// Gets the link data part reference using the specified link.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="link">
50     /// <para>The link.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>A ref raw link data part of t link</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink 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<TLink> GetHeaderReference() => ref *_header;
72 }
73 }

```

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

```

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

```

```

35    /// The header.
36    /// </para>
37    /// <para></para>
38    /// </summary>
39    protected new readonly LinksHeader<TLink>* Header;
40
41    /// <summary>
42    /// <para>
43    /// Initializes a new <see
44    ↪ cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45    /// </para>
46    /// <para></para>
47    /// </summary>
48    /// <param name="constants">
49    /// <para>A constants.</para>
50    /// <para></para>
51    /// </param>
52    /// <param name="linksDataParts">
53    /// <para>A links data parts.</para>
54    /// <para></para>
55    /// </param>
56    /// <param name="linksIndexParts">
57    /// <para>A links index parts.</para>
58    /// <para></para>
59    /// </param>
60    /// <param name="header">
61    /// <para>A header.</para>
62    /// <para></para>
63    /// </param>
64    [MethodImpl(MethodImplOptions.AggressiveInlining)]
65    protected
66    ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
67    ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
68    ↪ linksIndexParts, LinksHeader<TLink>* header)
69    : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
70    {
71    LinksDataParts = linksDataParts;
72    LinksIndexParts = linksIndexParts;
73    Header = header;
74    }
75
76    /// <summary>
77    /// <para>
78    /// Gets the zero.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <returns>
83    /// <para>The ulong</para>
84    /// <para></para>
85    /// </returns>
86    [MethodImpl(MethodImplOptions.AggressiveInlining)]
87    protected override ulong GetZero() => 0UL;
88
89    /// <summary>
90    /// <para>
91    /// Determines whether this instance equal to zero.
92    /// </para>
93    /// <para></para>
94    /// </summary>
95    /// <param name="value">
96    /// <para>The value.</para>
97    /// <para></para>
98    /// </param>
99    /// <returns>
100    /// <para>The bool</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected override bool EqualToZero(ulong value) => value == 0UL;
105
106    /// <summary>
107    /// <para>
108    /// Determines whether this instance are equal.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <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     /// <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     /// <summary>
215     /// <para>
216     /// Determines whether this instance less or equal than.
217     /// </para>
218     /// <para></para>
219     /// </summary>
220     /// <param name="first">
221     /// <para>The first.</para>
222     /// <para></para>
223     /// </param>
224     /// <param name="second">
225     /// <para>The second.</para>
226     /// <para></para>
227     /// </param>
228     /// <returns>
229     /// <para>The bool</para>
230     /// <para></para>
231     /// </returns>
232     [MethodImpl(MethodImplOptions.AggressiveInlining)]
233     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;

234     /// <summary>
235     /// <para>
236     /// Determines whether this instance less than zero.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="value">
241     /// <para>The value.</para>
242     /// <para></para>
243     /// </param>
244     /// <returns>
245     /// <para>The bool</para>
246     /// <para></para>
247     /// </returns>
248     [MethodImpl(MethodImplOptions.AggressiveInlining)]
249     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪ for ulong

250     /// <summary>
251     /// <para>
252     /// Determines whether this instance less than.
253     /// </para>
254     /// <para></para>
255     /// </summary>
256     /// <param name="first">
257     /// <para>The first.</para>
258     /// <para></para>
259     /// </param>
260     /// <param name="second">
261     /// <para>The second.</para>
262     /// <para></para>
263     /// </param>
264     /// <returns>
265     /// <para>The bool</para>
266     /// <para></para>
267     /// </returns>
268     [MethodImpl(MethodImplOptions.AggressiveInlining)]
269     protected override bool LessThan(ulong first, ulong second) => first < second;

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

39     protected new readonly LinksHeader<TLink>* Header;
40
41     /// <summary>
42     /// <para>
43     /// Initializes a new <see cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
44     ↪ instance.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="constants">
49     /// <para>A constants.</para>
50     /// <para></para>
51     /// </param>
52     /// <param name="linksDataParts">
53     /// <para>A links data parts.</para>
54     /// <para></para>
55     /// </param>
56     /// <param name="linksIndexParts">
57     /// <para>A links index parts.</para>
58     /// <para></para>
59     /// </param>
60     /// <param name="header">
61     /// <para>A header.</para>
62     /// <para></para>
63     /// </param>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected UInt64ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
66     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
67     ↪ linksIndexParts, LinksHeader<TLink>* header)
68     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
69     {
70         LinksDataParts = linksDataParts;
71         LinksIndexParts = linksIndexParts;
72         Header = header;
73     }
74
75     /// <summary>
76     /// <para>
77     /// Gets the zero.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <returns>
82     /// <para>The ulong</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override ulong GetZero() => OUL;
87
88     /// <summary>
89     /// <para>
90     /// Determines whether this instance equal to zero.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="value">
95     /// <para>The value.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The bool</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override bool EqualToZero(ulong value) => value == OUL;
104
105    /// <summary>
106    /// <para>
107    /// Determines whether this instance are equal.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="first">
112    /// <para>The first.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="second">
116    /// <para>The second.</para>

```

```

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

```

```

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

```

```

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

```



```

345 /// <para>The ulong</para>
346 /// <para></para>
347 /// </returns>
348 [MethodImpl(MethodImplOptions.AggressiveInlining)]
349 protected override ulong Subtract(ulong first, ulong second) => first - second;
350
351 /// <summary>
352 /// <para>
353 /// Gets the header reference.
354 /// </para>
355 /// <para></para>
356 /// </summary>
357 /// <returns>
358 /// <para>A ref links header of t link</para>
359 /// <para></para>
360 /// </returns>
361 [MethodImpl(MethodImplOptions.AggressiveInlining)]
362 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;
363
364 /// <summary>
365 /// <para>
366 /// Gets the link data part reference using the specified link.
367 /// </para>
368 /// <para></para>
369 /// </summary>
370 /// <param name="link">
371 /// <para>The link.</para>
372 /// <para></para>
373 /// </param>
374 /// <returns>
375 /// <para>A ref raw link data part of t link</para>
376 /// <para></para>
377 /// </returns>
378 [MethodImpl(MethodImplOptions.AggressiveInlining)]
379 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
380     ↪ ref LinksDataParts[link];
381
382 /// <summary>
383 /// <para>
384 /// Gets the link index part reference using the specified link.
385 /// </para>
386 /// <para></para>
387 /// </summary>
388 /// <param name="link">
389 /// <para>The link.</para>
390 /// <para></para>
391 /// </param>
392 /// <returns>
393 /// <para>A ref raw link index part of t link</para>
394 /// <para></para>
395 /// </returns>
396 [MethodImpl(MethodImplOptions.AggressiveInlining)]
397 protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
398     ↪ ref LinksIndexParts[link];
399
400 /// <summary>
401 /// <para>
402 /// Determines whether this instance first is to the left of second.
403 /// </para>
404 /// <para></para>
405 /// </summary>
406 /// <param name="first">
407 /// <para>The first.</para>
408 /// <para></para>
409 /// </param>
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(TLink first, TLink 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(TLink first, TLink second)
444     {
445         ref var firstLink = ref LinksDataParts[first];
446         ref var secondLink = ref LinksDataParts[second];
447         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
448             ↪ secondLink.Source, secondLink.Target);
449     }
450 }

```

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

```

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

```

```

40 public
    ↳ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↳ linksIndexParts, header) { }
41
42 /// <summary>
43 /// <para>
44 /// Gets the left reference using the specified node.
45 /// </para>
46 /// <para></para>
47 /// </summary>
48 /// <param name="node">
49 /// <para>The node.</para>
50 /// <para></para>
51 /// </param>
52 /// <returns>
53 /// <para>The ref link</para>
54 /// <para></para>
55 /// </returns>
56 [MethodImpl(MethodImplOptions.AggressiveInlining)]
57 protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ LinksIndexParts[node].LeftAsSource;
58
59 /// <summary>
60 /// <para>
61 /// Gets the right reference using the specified node.
62 /// </para>
63 /// <para></para>
64 /// </summary>
65 /// <param name="node">
66 /// <para>The node.</para>
67 /// <para></para>
68 /// </param>
69 /// <returns>
70 /// <para>The ref link</para>
71 /// <para></para>
72 /// </returns>
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 protected override ref TLink GetRightReference(TLink node) => ref
    ↳ LinksIndexParts[node].RightAsSource;
75
76 /// <summary>
77 /// <para>
78 /// Gets the left using the specified node.
79 /// </para>
80 /// <para></para>
81 /// </summary>
82 /// <param name="node">
83 /// <para>The node.</para>
84 /// <para></para>
85 /// </param>
86 /// <returns>
87 /// <para>The link</para>
88 /// <para></para>
89 /// </returns>
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93 /// <summary>
94 /// <para>
95 /// Gets the right using the specified node.
96 /// </para>
97 /// <para></para>
98 /// </summary>
99 /// <param name="node">
100 /// <para>The node.</para>
101 /// <para></para>
102 /// </param>
103 /// <returns>
104 /// <para>The link</para>
105 /// <para></para>
106 /// </returns>
107 [MethodImpl(MethodImplOptions.AggressiveInlining)]
108 protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
109
110 /// <summary>
111 /// <para>

```

```

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

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 external links sources size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64ExternalLinksSourcesSizeBalancedTreeMethods :
16         ↪ UInt64ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt64ExternalLinksSourcesSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt64ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
43             ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44             ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45             ↪ linksIndexParts, header) { }
46
47         /// <summary>
48         /// <para>
49         /// Gets the left reference using the specified node.
50         /// </para>

```

```

46     /// <para></para>
47     /// </summary>
48     /// <param name="node">
49     /// <para>The node.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>The ref link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLink GetLeftReference(TLink node) => ref
    ↪ LinksIndexParts[node].LeftAsSource;

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

75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsSource;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>

```

```

122     /// <para></para>
123     /// </param>
124     [MethodImpl(MethodImplOptions.AggressiveInlining)]
125     protected override void SetLeft(TLink node, TLink left) =>
126         ↳ LinksIndexParts[node].LeftAsSource = left;
127
128     /// <summary>
129     /// <para>
130     /// Sets the right using the specified node.
131     /// </para>
132     /// </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(TLink node, TLink right) =>
141         ↳ LinksIndexParts[node].RightAsSource = right;
142
143     /// <summary>
144     /// <para>
145     /// Gets the size using the specified node.
146     /// </para>
147     /// </summary>
148     /// <param name="node">
149     /// <para>The node.</para>
150     /// </param>
151     /// <returns>
152     /// <para>The link</para>
153     /// </returns>
154     [MethodImpl(MethodImplOptions.AggressiveInlining)]
155     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
156
157     /// <summary>
158     /// <para>
159     /// Sets the size using the specified node.
160     /// </para>
161     /// </summary>
162     /// <param name="node">
163     /// <para>The node.</para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// </param>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     protected override void SetSize(TLink node, TLink size) =>
170         ↳ LinksIndexParts[node].SizeAsSource = size;
171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// </summary>
177     /// <returns>
178     /// <para>The link</para>
179     /// </returns>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     protected override TLink GetTreeRoot() => Header->RootAsSource;
182
183     /// <summary>
184     /// <para>
185     /// Gets the base part value using the specified node.
186     /// </para>
187     /// </summary>

```



```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 external links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16     ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public
43         ↪ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
44         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
45         ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
46         ↪ linksIndexParts, header) { }
47
48         /// <summary>
49         /// <para>
50         /// Gets the left reference using the specified node.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="node">
55         /// <para>The node.</para>
56         /// <para></para>
57         /// </param>
58         /// <returns>
59         /// <para>The ref link</para>
60         /// <para></para>

```

```

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

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

75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
    ↪ LinksIndexParts[node].LeftAsTarget = left;

126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.

```

```

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

```

```

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

```

```

280     {
281         ref var link = ref LinksIndexParts[node];
282         link.LeftAsTarget = Zero;
283         link.RightAsTarget = Zero;
284         link.SizeAsTarget = Zero;
285     }
286 }
287 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 external links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64ExternalLinksTargetsSizeBalancedTreeMethods :
16     ↪ UInt64ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt64ExternalLinksTargetsSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public UInt64ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
43         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44         ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45         ↪ linksIndexParts, header) { }
46
47         /// <summary>
48         /// <para>
49         /// Gets the left reference using the specified node.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="node">
54         /// <para>The node.</para>
55         /// <para></para>
56         /// </param>
57         /// <returns>
58         /// <para>The ref link</para>
59         /// <para></para>
60         /// </returns>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         protected override ref TLink GetLeftReference(TLink node) => ref
63         ↪ LinksIndexParts[node].LeftAsTarget;
64
65         /// <summary>
66         /// <para>
67         /// Gets the right reference using the specified node.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         /// <param name="node">
72         /// <para>The node.</para>
73         /// <para></para>
74         /// </param>
75         /// <returns>
76         /// <para>The ref link</para>
77         /// <para></para>
78         /// </returns>
79         [MethodImpl(MethodImplOptions.AggressiveInlining)]
80         protected override ref TLink GetRightReference(TLink node) => ref
81         ↪ LinksIndexParts[node].RightAsTarget;
82     }
83 }

```

```

64     /// </summary>
65     /// <param name="node">
66     /// <para>The node.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The ref link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref TLink GetRightReference(TLink node) => ref
75     ↪ LinksIndexParts[node].RightAsTarget;
76
77     /// <summary>
78     /// <para>
79     /// Gets the left using the specified node.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="node">
84     /// <para>The node.</para>
85     /// <para></para>
86     /// </param>
87     /// <returns>
88     /// <para>The link</para>
89     /// <para></para>
90     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected override TLink GetLeft(TLink node) => 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 TLink GetRight(TLink node) => 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(TLink node, TLink 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>

```

```

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

```



```

216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236         ↪ TLink secondSource, TLink secondTarget)
237         => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
238         ↪ secondSource;
239
240     /// <summary>
241     /// <para>
242     /// Determines whether this instance first is to the right of second.
243     /// </para>
244     /// <para></para>
245     /// </summary>
246     /// <param name="firstSource">
247     /// <para>The first source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="firstTarget">
251     /// <para>The first target.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondSource">
255     /// <para>The second source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="secondTarget">
259     /// <para>The second target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The bool</para>
264     /// <para></para>
265     /// </returns>
266     [MethodImpl(MethodImplOptions.AggressiveInlining)]
267     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
268         ↪ TLink secondSource, TLink secondTarget)
269         => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
270         ↪ secondSource;
271
272     /// <summary>
273     /// <para>
274     /// Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override void ClearNode(TLink node)
284     {
285         ref var link = ref LinksIndexParts[node];
286         link.LeftAsTarget = Zero;
287         link.RightAsTarget = Zero;
288         link.SizeAsTarget = Zero;
289     }
290 }
291 }

```

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

```
1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLink = System.UInt64;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 64 internal links recursionless size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
16    public unsafe abstract class UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
18    {
19        /// <summary>
20        /// <para>
21        /// The links data parts.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26        /// <summary>
27        /// <para>
28        /// The links index parts.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33        /// <summary>
34        /// <para>
35        /// The header.
36        /// </para>
37        /// <para></para>
38        /// </summary>
39        protected new readonly LinksHeader<TLink>* Header;
40
41        /// <summary>
42        /// <para>
43        /// Initializes a new <see
44        ↪ cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
45        /// </para>
46        /// <para></para>
47        /// </summary>
48        /// <param name="constants">
49        /// <para>A constants.</para>
50        /// <para></para>
51        /// </param>
52        /// <param name="linksDataParts">
53        /// <para>A links data parts.</para>
54        /// <para></para>
55        /// </param>
56        /// <param name="linksIndexParts">
57        /// <para>A links index parts.</para>
58        /// <para></para>
59        /// </param>
60        /// <param name="header">
61        /// <para>A header.</para>
62        /// <para></para>
63        /// </param>
64        [MethodImpl(MethodImplOptions.AggressiveInlining)]
65        protected
66        ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
67        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
68        ↪ linksIndexParts, LinksHeader<TLink>* header)
69        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
70        {
71            LinksDataParts = linksDataParts;
72            LinksIndexParts = linksIndexParts;
73            Header = header;
74        }
75
76        /// <summary>
77        /// <para>
78        /// Gets the zero.
79        /// </para>
80        /// </summary>
81        public static TLink Zero;
```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 internal links size balanced tree methods base.
12     /// </para>
13     /// </summary>
14     /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
15     public unsafe abstract class UInt64InternalLinksSizeBalancedTreeMethodsBase :
16         ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
17     {
18         /// <summary>
19         /// <para>
20         /// The links data parts.
21         /// </para>
22         /// </summary>
23         protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
24         /// <summary>
25         /// <para>
26         /// The links index parts.
27         /// </para>

```

```

28     /// </para>
29     /// <para></para>
30     /// </summary>
31     protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
32     /// <summary>
33     /// <para>
34     /// The header.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     protected new readonly LinksHeader<TLink>* Header;
39
40     /// <summary>
41     /// <para>
42     /// Initializes a new <see cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
43     ↪ instance.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="constants">
48     /// <para>A constants.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="linksDataParts">
52     /// <para>A links data parts.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="linksIndexParts">
56     /// <para>A links index parts.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="header">
60     /// <para>A header.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected UInt64InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
65     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
66     ↪ linksIndexParts, LinksHeader<TLink>* header)
67     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
68     {
69         LinksDataParts = linksDataParts;
70         LinksIndexParts = linksIndexParts;
71         Header = header;
72     }
73
74     /// <summary>
75     /// <para>
76     /// Gets the zero.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetZero() => 0UL;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance equal to zero.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="value">
94     /// <para>The value.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The bool</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override bool EqualToZero(ulong value) => value == 0UL;
103
104    /// <summary>
105    /// <para>

```



```

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

```

```

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

```

```

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

```

```

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

```

```

409     /// <para></para>
410     /// </summary>
411     /// <param name="first">
412     /// <para>The first.</para>
413     /// <para></para>
414     /// </param>
415     /// <param name="second">
416     /// <para>The second.</para>
417     /// <para></para>
418     /// </param>
419     /// <returns>
420     /// <para>The bool</para>
421     /// <para></para>
422     /// </returns>
423     [MethodImpl(MethodImplOptions.AggressiveInlining)]
424     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
        ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
425 }
426 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Generic
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links sources linked list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLink}" />
15     public unsafe class UInt64InternalLinksSourcesLinkedListMethods :
        ↪ InternalLinksSourcesLinkedListMethods<TLink>
16     {
17         private readonly RawLinkDataPart<TLink>* _linksDataParts;
18         private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="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<TLink> constants,
        ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
40         {
41             _linksDataParts = linksDataParts;
42             _linksIndexParts = linksIndexParts;
43         }
44
45         /// <summary>
46         /// <para>
47         /// Gets the link data part reference using the specified link.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="link">
52         /// <para>The link.</para>
53         /// <para></para>
54         /// </param>
55     }

```

```

56     /// <returns>
57     /// <para>A ref raw link data part of t link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪ ref _linksDataParts[link];
62
63     /// <summary>
64     /// <para>
65     /// Gets the link index part reference using the specified link.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="link">
70     /// <para>The link.</para>
71     /// <para></para>
72     /// </param>
73     /// <returns>
74     /// <para>A ref raw link index part of t link</para>
75     /// <para></para>
76     /// </returns>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪ ref _linksIndexParts[link];
79 }
80 }

```

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

```

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

```

```

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

```

```

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

```



```

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

```

```

267     }
268 }

```

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

```

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

```

```

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

```

```

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

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10     /// Represents the int 64 internal links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16         ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
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     }
34 }

```

```

29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public
41     ↪ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
42     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
43     ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
44     ↪ linksIndexParts, header) { }
45
46     /// <summary>
47     /// <para>
48     /// Gets the left reference using the specified node.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="node">
53     /// <para>The node.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>The ref ulong</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref ulong GetLeftReference(ulong node) => ref
62     ↪ LinksIndexParts[node].LeftAsTarget;
63
64     /// <summary>
65     /// <para>
66     /// Gets the right reference using the specified node.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="node">
71     /// <para>The node.</para>
72     /// <para></para>
73     /// </param>
74     /// <returns>
75     /// <para>The ref ulong</para>
76     /// <para></para>
77     /// </returns>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected override ref ulong GetRightReference(ulong node) => ref
80     ↪ LinksIndexParts[node].RightAsTarget;
81
82     /// <summary>
83     /// <para>
84     /// Gets the left using the specified node.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="node">
89     /// <para>The node.</para>
90     /// <para></para>
91     /// </param>
92     /// <returns>
93     /// <para>The link</para>
94     /// <para></para>
95     /// </returns>
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
98
99     /// <summary>
100    /// <para>
101    /// Gets the right using the specified node.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    /// <param name="node">
106    /// <para>The node.</para>
107    /// <para></para>
108    /// </param>
109    /// <returns>
110    /// <para>The link</para>
111    /// <para></para>
112    /// </returns>
113    [MethodImpl(MethodImplOptions.AggressiveInlining)]
114    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;

```

```

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

```

```

176     protected override void SetSize(TLink node, TLink size) =>
177         ↳ LinksIndexParts[node].SizeAsTarget = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root using the specified node.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="node">
186     /// <para>The node.</para>
187     /// <para></para>
188     /// </param>
189     /// <returns>
190     /// <para>The link</para>
191     /// <para></para>
192     /// </returns>
193     [MethodImpl(MethodImplOptions.AggressiveInlining)]
194     protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsTarget;
195
196     /// <summary>
197     /// <para>
198     /// Gets the base part value using the specified node.
199     /// </para>
200     /// <para></para>
201     /// </summary>
202     /// <param name="node">
203     /// <para>The node.</para>
204     /// <para></para>
205     /// </param>
206     /// <returns>
207     /// <para>The link</para>
208     /// <para></para>
209     /// </returns>
210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
211     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
212
213     /// <summary>
214     /// <para>
215     /// Gets the key part value using the specified node.
216     /// </para>
217     /// <para></para>
218     /// </summary>
219     /// <param name="node">
220     /// <para>The node.</para>
221     /// <para></para>
222     /// </param>
223     /// <returns>
224     /// <para>The link</para>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     protected override TLink GetKeyPartValue(TLink node) => 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(TLink 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>

```



```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksTargetsSizeBalancedTreeMethods :
        ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UInt64InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
        ↪ linksIndexParts, header) { }
41
42         /// <summary>
43         /// <para>
44         /// Gets the left reference using the specified node.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="node">
49         /// <para>The node.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>The ref ulong</para>
54         /// <para></para>

```

```

55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref ulong GetLeftReference(ulong node) => ref
    ↪ 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 TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110    /// <summary>
111    /// <para>
112    /// Sets the left using the specified node.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="node">
117    /// <para>The node.</para>
118    /// <para></para>
119    /// </param>
120    /// <param name="left">
121    /// <para>The left.</para>
122    /// <para></para>
123    /// </param>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected override void SetLeft(TLink node, TLink left) =>
    ↪ LinksIndexParts[node].LeftAsTarget = left;

126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.

```

```

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

```

```

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

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLink = System.UInt64;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>

```

```

14  /// Represents the int 64 split memory links.
15  /// </para>
16  /// <para></para>
17  /// </summary>
18  /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
19  public unsafe class UInt64SplitMemoryLinks : SplitMemoryLinksBase<TLink>
20  {
21      private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
22      private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
23      private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
24      private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
25      private LinksHeader<ulong>* _header;
26      private RawLinkDataPart<ulong>* _linksDataParts;
27      private RawLinkIndexPart<ulong>* _linksIndexParts;
28
29      /// <summary>
30      /// <para>
31      /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
32      /// </para>
33      /// <para></para>
34      /// </summary>
35      /// <param name="dataMemory">
36      /// <para>A data memory.</para>
37      /// <para></para>
38      /// </param>
39      /// <param name="indexMemory">
40      /// <para>A index memory.</para>
41      /// <para></para>
42      /// </param>
43      [MethodImpl(MethodImplOptions.AggressiveInlining)]
44      public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ 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<TLink>>.Instance,
        ↳ IndexTreeType.Default, useLinkedList: true) { }
66
67      /// <summary>
68      /// <para>
69      /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
70      /// </para>
71      /// <para></para>
72      /// </summary>
73      /// <param name="dataMemory">
74      /// <para>A data memory.</para>
75      /// <para></para>
76      /// </param>
77      /// <param name="indexMemory">
78      /// <para>A index memory.</para>
79      /// <para></para>
80      /// </param>
81      /// <param name="memoryReservationStep">
82      /// <para>A memory reservation step.</para>
83      /// <para></para>
84      /// </param>
85      /// <param name="constants">
86      /// <para>A constants.</para>
87      /// <para></para>

```

```

88     /// </param>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪     indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
    ↪     this(dataMemory, indexMemory, memoryReservationStep, constants,
    ↪     IndexTreeType.Default, useLinkedList: true) { }

91
92     /// <summary>
93     /// <para>
94     /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <param name="dataMemory">
99     /// <para>A data memory.</para>
100    /// <para></para>
101    /// </param>
102    /// <param name="indexMemory">
103    /// <para>A index memory.</para>
104    /// <para></para>
105    /// </param>
106    /// <param name="memoryReservationStep">
107    /// <para>A memory reservation step.</para>
108    /// <para></para>
109    /// </param>
110    /// <param name="constants">
111    /// <para>A constants.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="indexTreeType">
115    /// <para>A index tree type.</para>
116    /// <para></para>
117    /// </param>
118    /// <param name="useLinkedList">
119    /// <para>A use linked list.</para>
120    /// <para></para>
121    /// </param>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪     indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
    ↪     IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
    ↪     memoryReservationStep, constants, useLinkedList)
124    {
125        if (indexTreeType == IndexTreeType.SizeBalancedTree)
126        {
127            _createInternalSourceTreeMethods = () => new
    ↪            UInt64InternalLinksSourcesSizeBalancedTreeMethods(Constants,
    ↪            _linksDataParts, _linksIndexParts, _header);
128            _createExternalSourceTreeMethods = () => new
    ↪            UInt64ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
    ↪            _linksDataParts, _linksIndexParts, _header);
129            _createInternalTargetTreeMethods = () => new
    ↪            UInt64InternalLinksTargetsSizeBalancedTreeMethods(Constants,
    ↪            _linksDataParts, _linksIndexParts, _header);
130            _createExternalTargetTreeMethods = () => new
    ↪            UInt64ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
    ↪            _linksDataParts, _linksIndexParts, _header);
131        }
132        else
133        {
134            _createInternalSourceTreeMethods = () => new
    ↪            UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
    ↪            _linksDataParts, _linksIndexParts, _header);
135            _createExternalSourceTreeMethods = () => new
    ↪            UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
    ↪            _linksDataParts, _linksIndexParts, _header);
136            _createInternalTargetTreeMethods = () => new
    ↪            UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
    ↪            _linksDataParts, _linksIndexParts, _header);
137            _createExternalTargetTreeMethods = () => new
    ↪            UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
    ↪            _linksDataParts, _linksIndexParts, _header);
138        }
139        Init(dataMemory, indexMemory);
140    }
141
142    /// <summary>

```

```

143 /// <para>
144 /// Sets the pointers using the specified data memory.
145 /// </para>
146 /// <para></para>
147 /// </summary>
148 /// <param name="dataMemory">
149 /// <para>The data memory.</para>
150 /// <para></para>
151 /// </param>
152 /// <param name="indexMemory">
153 /// <para>The index memory.</para>
154 /// <para></para>
155 /// </param>
156 [MethodImpl(MethodImplOptions.AggressiveInlining)]
157 protected override void SetPointers(IResizableDirectMemory dataMemory,
158   ↪ IResizableDirectMemory indexMemory)
159 {
160     _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
161     _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
162     _header = (LinksHeader<TLink>*)indexMemory.Pointer;
163     if (_useLinkedList)
164     {
165         InternalSourcesListMethods = new
166             ↪ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
167             ↪ _linksIndexParts);
168     }
169     else
170     {
171         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
172     }
173     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
174     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
175     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
176     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_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<TLink> 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<TLink> GetLinkDataPartReference(TLink 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<TLink> GetLinkIndexPartReference(TLink
    ↳ 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(ulong first, ulong second) => first == second;
258
259 /// <summary>
260 /// <para>
261 /// Determines whether this instance less than.
262 /// </para>
263 /// <para></para>
264 /// </summary>
265 /// <param name="first">
266 /// <para>The first.</para>
267 /// <para></para>
268 /// </param>
269 /// <param name="second">
270 /// <para>The second.</para>
271 /// <para></para>
272 /// </param>
273 /// <returns>
274 /// <para>The bool</para>
275 /// <para></para>
276 /// </returns>
277 [MethodImpl(MethodImplOptions.AggressiveInlining)]
278 protected override bool LessThan(ulong first, ulong second) => first < second;
279
280 /// <summary>
281 /// <para>
282 /// Determines whether this instance less or equal than.
283 /// </para>
284 /// <para></para>
285 /// </summary>
286 /// <param name="first">
287 /// <para>The first.</para>
288 /// <para></para>
289 /// </param>
290 /// <param name="second">
291 /// <para>The second.</para>
292 /// <para></para>
293 /// </param>

```



```

294     /// <returns>
295     /// <para>The bool</para>
296     /// <para></para>
297     /// </returns>
298     [MethodImpl(MethodImplOptions.AggressiveInlining)]
299     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
300
301     /// <summary>
302     /// <para>
303     /// Determines whether this instance greater than.
304     /// </para>
305     /// <para></para>
306     /// </summary>
307     /// <param name="first">
308     /// <para>The first.</para>
309     /// <para></para>
310     /// </param>
311     /// <param name="second">
312     /// <para>The second.</para>
313     /// <para></para>
314     /// </param>
315     /// <returns>
316     /// <para>The bool</para>
317     /// <para></para>
318     /// </returns>
319     [MethodImpl(MethodImplOptions.AggressiveInlining)]
320     protected override bool GreaterThan(ulong first, ulong second) => first > second;
321
322     /// <summary>
323     /// <para>
324     /// Determines whether this instance greater or equal than.
325     /// </para>
326     /// <para></para>
327     /// </summary>
328     /// <param name="first">
329     /// <para>The first.</para>
330     /// <para></para>
331     /// </param>
332     /// <param name="second">
333     /// <para>The second.</para>
334     /// <para></para>
335     /// </param>
336     /// <returns>
337     /// <para>The bool</para>
338     /// <para></para>
339     /// </returns>
340     [MethodImpl(MethodImplOptions.AggressiveInlining)]
341     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
342
343     /// <summary>
344     /// <para>
345     /// Gets the zero.
346     /// </para>
347     /// <para></para>
348     /// </summary>
349     /// <returns>
350     /// <para>The ulong</para>
351     /// <para></para>
352     /// </returns>
353     [MethodImpl(MethodImplOptions.AggressiveInlining)]
354     protected override ulong GetZero() => 0UL;
355
356     /// <summary>
357     /// <para>
358     /// Gets the one.
359     /// </para>
360     /// <para></para>
361     /// </summary>
362     /// <returns>
363     /// <para>The ulong</para>
364     /// <para></para>
365     /// </returns>
366     [MethodImpl(MethodImplOptions.AggressiveInlining)]
367     protected override ulong GetOne() => 1UL;
368
369     /// <summary>
370     /// <para>
371     /// Converts the to int 64 using the specified value.

```

```

372    /// </para>
373    /// <para></para>
374    /// </summary>
375    /// <param name="value">
376    /// <para>The value.</para>
377    /// <para></para>
378    /// </param>
379    /// <returns>
380    /// <para>The long</para>
381    /// <para></para>
382    /// </returns>
383    [MethodImpl(MethodImplOptions.AggressiveInlining)]
384    protected override long ConvertToInt64(ulong value) => (long)value;
385
386    /// <summary>
387    /// <para>
388    /// Converts the to address using the specified value.
389    /// </para>
390    /// <para></para>
391    /// </summary>
392    /// <param name="value">
393    /// <para>The value.</para>
394    /// <para></para>
395    /// </param>
396    /// <returns>
397    /// <para>The ulong</para>
398    /// <para></para>
399    /// </returns>
400    [MethodImpl(MethodImplOptions.AggressiveInlining)]
401    protected override ulong ConvertToAddress(long value) => (ulong)value;
402
403    /// <summary>
404    /// <para>
405    /// Adds the first.
406    /// </para>
407    /// <para></para>
408    /// </summary>
409    /// <param name="first">
410    /// <para>The first.</para>
411    /// <para></para>
412    /// </param>
413    /// <param name="second">
414    /// <para>The second.</para>
415    /// <para></para>
416    /// </param>
417    /// <returns>
418    /// <para>The ulong</para>
419    /// <para></para>
420    /// </returns>
421    [MethodImpl(MethodImplOptions.AggressiveInlining)]
422    protected override ulong Add(ulong first, ulong second) => first + second;
423
424    /// <summary>
425    /// <para>
426    /// Subtracts the first.
427    /// </para>
428    /// <para></para>
429    /// </summary>
430    /// <param name="first">
431    /// <para>The first.</para>
432    /// <para></para>
433    /// </param>
434    /// <param name="second">
435    /// <para>The second.</para>
436    /// <para></para>
437    /// </param>
438    /// <returns>
439    /// <para>The ulong</para>
440    /// <para></para>
441    /// </returns>
442    [MethodImpl(MethodImplOptions.AggressiveInlining)]
443    protected override ulong Subtract(ulong first, ulong second) => first - second;
444
445    /// <summary>
446    /// <para>
447    /// Increments the link.
448    /// </para>
449    /// <para></para>

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 unused links list methods.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="UnusedLinksListMethods{TLink}" />
16     public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<TLink>
17     {
18         private readonly RawLinkDataPart<ulong>* _links;
19         private readonly LinksHeader<ulong>* _header;
20
21         /// <summary>
22         /// <para>
23         /// Initializes a new <see cref="UInt64UnusedLinksListMethods" /> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt64UnusedLinksListMethods(RawLinkDataPart<ulong>* links, LinksHeader<ulong>*
37             ↪ header)
38             : base((byte*)links, (byte*)header)
39         {
40             _links = links;
41             _header = header;
42         }
43
44         /// <summary>
45         /// <para>
46         /// Gets the link data part reference using the specified link.
47         /// </para>

```

```

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

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using Platform.Numbers;
9  using static System.Runtime.CompilerServices.Unsafe;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.Memory.United.Generic
14 {
15     /// <summary>
16     /// <para>
17     /// Represents the links avl balanced tree methods base.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     /// <seealso cref="SizedAndThreadedAVLBalancedTreeMethods{TLink}"/>
22     /// <seealso cref="ILinksTreeMethods{TLink}"/>
23     public unsafe abstract class LinksAvlBalancedTreeMethodsBase<TLink> :
24         ↪ SizedAndThreadedAVLBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
25     {
26         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
27             ↪ UncheckedConverter<TLink, long>.Default;
28         private static readonly UncheckedConverter<TLink, int> _addressToInt32Converter =
29             ↪ UncheckedConverter<TLink, int>.Default;
30         private static readonly UncheckedConverter<bool, TLink> _boolToAddressConverter =
31             ↪ UncheckedConverter<bool, TLink>.Default;
32         private static readonly UncheckedConverter<TLink, bool> _addressToBoolConverter =
33             ↪ UncheckedConverter<TLink, bool>.Default;
34         private static readonly UncheckedConverter<int, TLink> _int32ToAddressConverter =
35             ↪ UncheckedConverter<int, TLink>.Default;
36
37         /// <summary>
38         /// <para>
39         /// The break.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         protected readonly TLink Break;
44
45         /// <summary>
46         /// <para>
47         /// The continue.
48         /// </para>
49         /// <para></para>
50         /// </summary>

```

```

44     protected readonly TLink Continue;
45     /// <summary>
46     /// <para>
47     /// The links.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     protected readonly byte* Links;
52     /// <summary>
53     /// <para>
54     /// The header.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     protected readonly byte* Header;
59
60     /// <summary>
61     /// <para>
62     /// Initializes a new <see cref="LinksAvlBalancedTreeMethodsBase"/> instance.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="constants">
67     /// <para>A constants.</para>
68     /// <para></para>
69     /// </param>
70     /// <param name="links">
71     /// <para>A links.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="header">
75     /// <para>A header.</para>
76     /// <para></para>
77     /// </param>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected LinksAvlBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
80     ↪ byte* header)
81     {
82         Links = links;
83         Header = header;
84         Break = constants.Break;
85         Continue = constants.Continue;
86     }
87
88     /// <summary>
89     /// <para>
90     /// Gets the tree root.
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 abstract TLink GetTreeRoot();
100
101     /// <summary>
102     /// <para>
103     /// Gets the base part value using the specified link.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="link">
108     /// <para>The link.</para>
109     /// <para></para>
110     /// </param>
111     /// <returns>
112     /// <para>The link</para>
113     /// <para></para>
114     /// </returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     protected abstract TLink GetBasePartValue(TLink link);
117
118     /// <summary>
119     /// <para>
120     /// Determines whether this instance first is to the right of second.
121     /// </para>

```

```

121    /// <para></para>
122    /// </summary>
123    /// <param name="source">
124    /// <para>The source.</para>
125    /// <para></para>
126    /// </param>
127    /// <param name="target">
128    /// <para>The target.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="rootSource">
132    /// <para>The root source.</para>
133    /// <para></para>
134    /// </param>
135    /// <param name="rootTarget">
136    /// <para>The root target.</para>
137    /// <para></para>
138    /// </param>
139    /// <returns>
140    /// <para>The bool</para>
141    /// <para></para>
142    /// </returns>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);
145
146    /// <summary>
147    /// <para>
148    /// Determines whether this instance first is to the left of second.
149    /// </para>
150    /// <para></para>
151    /// </summary>
152    /// <param name="source">
153    /// <para>The source.</para>
154    /// <para></para>
155    /// </param>
156    /// <param name="target">
157    /// <para>The target.</para>
158    /// <para></para>
159    /// </param>
160    /// <param name="rootSource">
161    /// <para>The root source.</para>
162    /// <para></para>
163    /// </param>
164    /// <param name="rootTarget">
165    /// <para>The root target.</para>
166    /// <para></para>
167    /// </param>
168    /// <returns>
169    /// <para>The bool</para>
170    /// <para></para>
171    /// </returns>
172    [MethodImpl(MethodImplOptions.AggressiveInlining)]
173    protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);
174
175    /// <summary>
176    /// <para>
177    /// Gets the header reference.
178    /// </para>
179    /// <para></para>
180    /// </summary>
181    /// <returns>
182    /// <para>A ref links header of t link</para>
183    /// <para></para>
184    /// </returns>
185    [MethodImpl(MethodImplOptions.AggressiveInlining)]
186    protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLink>>(Header);
187
188    /// <summary>
189    /// <para>
190    /// Gets the link reference using the specified link.
191    /// </para>
192    /// <para></para>
193    /// </summary>
194    /// <param name="link">
195    /// <para>The link.</para>

```

```

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

```

```

271 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
272 {
273     ref var firstLink = ref GetLinkReference(first);
274     ref var secondLink = ref GetLinkReference(second);
275     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
276 }
277
278 /// <summary>
279 /// <para>
280 /// Gets the size value using the specified value.
281 /// </para>
282 /// <para></para>
283 /// </summary>
284 /// <param name="value">
285 /// <para>The value.</para>
286 /// <para></para>
287 /// </param>
288 /// <returns>
289 /// <para>The link</para>
290 /// <para></para>
291 /// </returns>
292 [MethodImpl(MethodImplOptions.AggressiveInlining)]
293 protected virtual TLink GetSizeValue(TLink value) => Bit<TLink>.PartialRead(value, 5,
    ↪ -5);
294
295 /// <summary>
296 /// <para>
297 /// Sets the size value using the specified stored value.
298 /// </para>
299 /// <para></para>
300 /// </summary>
301 /// <param name="storedValue">
302 /// <para>The stored value.</para>
303 /// <para></para>
304 /// </param>
305 /// <param name="size">
306 /// <para>The size.</para>
307 /// <para></para>
308 /// </param>
309 [MethodImpl(MethodImplOptions.AggressiveInlining)]
310 protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
    ↪ Bit<TLink>.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(TLink value)
328 {
329     unchecked
330     {
331         return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 4, 1));
332         //return !EqualityComparer.Equals(Bit<TLink>.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>
349 /// </param>
350 [MethodImpl(MethodImplOptions.AggressiveInlining)]
351 protected virtual void SetLeftIsChildValue(ref TLink storedValue, bool value)
352 {
353     unchecked
354     {
355         var previousValue = storedValue;
356         var modified = Bit<TLink>.PartialWrite(previousValue,
357             ↪ _boolToAddressConverter.Convert(value), 4, 1);
358         storedValue = modified;
359     }
360 }
361 /// <summary>
362 /// <para>
363 /// Determines whether this instance get right is child value.
364 /// </para>
365 /// </para>
366 /// </summary>
367 /// <param name="value">
368 /// <para>The value.</para>
369 /// </para>
370 /// </param>
371 /// <returns>
372 /// <para>The bool</para>
373 /// </para>
374 /// </returns>
375 [MethodImpl(MethodImplOptions.AggressiveInlining)]
376 protected virtual bool GetRightIsChildValue(TLink value)
377 {
378     unchecked
379     {
380         return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 3, 1));
381         //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 3, 1), default);
382     }
383 }
384 /// <summary>
385 /// <para>
386 /// Sets the right is child value using the specified stored value.
387 /// </para>
388 /// </para>
389 /// </summary>
390 /// <param name="storedValue">
391 /// <para>The stored value.</para>
392 /// </para>
393 /// </param>
394 /// <param name="value">
395 /// <para>The value.</para>
396 /// </para>
397 /// </param>
398 [MethodImpl(MethodImplOptions.AggressiveInlining)]
399 protected virtual void SetRightIsChildValue(ref TLink storedValue, bool value)
400 {
401     unchecked
402     {
403         var previousValue = storedValue;
404         var modified = Bit<TLink>.PartialWrite(previousValue,
405             ↪ _boolToAddressConverter.Convert(value), 3, 1);
406         storedValue = modified;
407     }
408 }
409 /// <summary>
410 /// <para>
411 /// Determines whether this instance is child.
412 /// </para>
413 /// </para>
414 /// </summary>
415 /// <param name="parent">
416 /// <para>The parent.</para>
417 /// </para>
418 /// </param>
419 /// <param name="possibleChild">
420 /// <para>The possible child.</para>

```

```

422     /// <para></para>
423     /// </param>
424     /// <returns>
425     /// <para>The bool</para>
426     /// <para></para>
427     /// </returns>
428     [MethodImpl(MethodImplOptions.AggressiveInlining)]
429     protected bool IsChild(TLink parent, TLink possibleChild)
430     {
431         var parentSize = GetSize(parent);
432         var childSize = GetSizeOrZero(possibleChild);
433         return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
434     }
435
436     /// <summary>
437     /// <para>
438     /// Gets the balance value using the specified stored value.
439     /// </para>
440     /// <para></para>
441     /// </summary>
442     /// <param name="storedValue">
443     /// <para>The stored value.</para>
444     /// <para></para>
445     /// </param>
446     /// <returns>
447     /// <para>The sbyte</para>
448     /// <para></para>
449     /// </returns>
450     [MethodImpl(MethodImplOptions.AggressiveInlining)]
451     protected virtual sbyte GetBalanceValue(TLink storedValue)
452     {
453         unchecked
454         {
455             var value = _addressToInt32Converter.Convert(Bit<TLink>.PartialRead(storedValue,
456                 ↪ 0, 3));
457             value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
458                 ↪ end of sbyte
459             return (sbyte)value;
460         }
461     }
462
463     /// <summary>
464     /// <para>
465     /// Sets the balance value using the specified stored value.
466     /// </para>
467     /// <para></para>
468     /// </summary>
469     /// <param name="storedValue">
470     /// <para>The stored value.</para>
471     /// <para></para>
472     /// </param>
473     /// <param name="value">
474     /// <para>The value.</para>
475     /// <para></para>
476     /// </param>
477     [MethodImpl(MethodImplOptions.AggressiveInlining)]
478     protected virtual void SetBalanceValue(ref TLink storedValue, sbyte value)
479     {
480         unchecked
481         {
482             var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
483                 ↪ value & 3);
484             var modified = Bit<TLink>.PartialWrite(storedValue, packagedValue, 0, 3);
485             storedValue = modified;
486         }
487     }
488
489     /// <summary>
490     /// <para>
491     /// The zero.
492     /// </para>
493     /// <para></para>
494     /// </summary>
495     public TLink this[TLink index]
496     {
497         [MethodImpl(MethodImplOptions.AggressiveInlining)]
498         get
499         {

```

```

497     var root = GetTreeRoot();
498     if (GreaterOrEqualThan(index, GetSize(root)))
499     {
500         return Zero;
501     }
502     while (!EqualToZero(root))
503     {
504         var left = GetLeftOrDefault(root);
505         var leftSize = GetSizeOrZero(left);
506         if (LessThan(index, leftSize))
507         {
508             root = left;
509             continue;
510         }
511         if (AreEqual(index, leftSize))
512         {
513             return root;
514         }
515         root = GetRightOrDefault(root);
516         index = Subtract(index, Increment(leftSize));
517     }
518     return Zero; // TODO: Impossible situation exception (only if tree structure
519                 ↪ broken)
520 }
521
522 /// <summary>
523 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
524 ↪ (концом).
525 /// </summary>
526 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
527 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
528 /// <returns>Индекс искомой связи.</returns>
529 [MethodImpl(MethodImplOptions.AggressiveInlining)]
530 public TLink Search(TLink source, TLink target)
531 {
532     var root = GetTreeRoot();
533     while (!EqualToZero(root))
534     {
535         ref var rootLink = ref GetLinkReference(root);
536         var rootSource = rootLink.Source;
537         var rootTarget = rootLink.Target;
538         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
539             ↪ node.Key < root.Key
540         {
541             root = GetLeftOrDefault(root);
542         }
543         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
544             ↪ node.Key > root.Key
545         {
546             root = GetRightOrDefault(root);
547         }
548         else // node.Key == root.Key
549         {
550             return root;
551         }
552     }
553     return Zero;
554 }
555
556 // TODO: Return indices range instead of references count
557 /// <summary>
558 /// <para>
559 /// Counts the usages using the specified link.
560 /// </para>
561 /// <para></para>
562 /// </summary>
563 /// <param name="link">
564 /// <para>The link.</para>
565 /// </param>
566 /// <returns>
567 /// <para>The link</para>
568 /// <para></para>
569 /// </returns>
570 [MethodImpl(MethodImplOptions.AggressiveInlining)]
571 public TLink CountUsages(TLink link)
572 {

```

```

571     var root = GetTreeRoot();
572     var total = GetSize(root);
573     var totalRightIgnore = Zero;
574     while (!EqualToZero(root))
575     {
576         var @base = GetBasePartValue(root);
577         if (LessOrEqualThan(@base, link))
578         {
579             root = GetRightOrDefault(root);
580         }
581         else
582         {
583             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
584             root = GetLeftOrDefault(root);
585         }
586     }
587     root = GetTreeRoot();
588     var totalLeftIgnore = Zero;
589     while (!EqualToZero(root))
590     {
591         var @base = GetBasePartValue(root);
592         if (GreaterOrEqualThan(@base, link))
593         {
594             root = GetLeftOrDefault(root);
595         }
596         else
597         {
598             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
599         }
600         root = GetRightOrDefault(root);
601     }
602 }
603 return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
604 }
605
606 /// <summary>
607 /// <para>
608 /// Eaches the usage using the specified link.
609 /// </para>
610 /// <para></para>
611 /// </summary>
612 /// <param name="link">
613 /// <para>The link.</para>
614 /// <para></para>
615 /// </param>
616 /// <param name="handler">
617 /// <para>The handler.</para>
618 /// <para></para>
619 /// </param>
620 /// <returns>
621 /// <para>The continue.</para>
622 /// <para></para>
623 /// </returns>
624 [MethodImpl(MethodImplOptions.AggressiveInlining)]
625 public TLink EachUsage(TLink link, ReadHandler<TLink>? handler)
626 {
627     var root = GetTreeRoot();
628     if (EqualToZero(root))
629     {
630         return Continue;
631     }
632     TLink first = Zero, current = root;
633     while (!EqualToZero(current))
634     {
635         var @base = GetBasePartValue(current);
636         if (GreaterOrEqualThan(@base, link))
637         {
638             if (AreEqual(@base, link))
639             {
640                 first = current;
641             }
642             current = GetLeftOrDefault(current);
643         }
644         else
645         {
646             current = GetRightOrDefault(current);
647         }
648     }

```

```

649         if (!EqualToZero(first))
650         {
651             current = first;
652             while (true)
653             {
654                 if (AreEqual(handler(GetLinkValues(current)), Break))
655                 {
656                     return Break;
657                 }
658                 current = GetNext(current);
659                 if (EqualToZero(current) || !AreEqual(GetBasePartValue(current), link))
660                 {
661                     break;
662                 }
663             }
664         }
665         return Continue;
666     }
667
668     /// <summary>
669     /// <para>
670     /// Prints the node value using the specified node.
671     /// </para>
672     /// <para></para>
673     /// </summary>
674     /// <param name="node">
675     /// <para>The node.</para>
676     /// <para></para>
677     /// </param>
678     /// <param name="sb">
679     /// <para>The sb.</para>
680     /// <para></para>
681     /// </param>
682     [MethodImpl(MethodImplOptions.AggressiveInlining)]
683     protected override void PrintNodeValue(TLink node, StringBuilder sb)
684     {
685         ref var link = ref GetLinkReference(node);
686         sb.Append(' ');
687         sb.Append(link.Source);
688         sb.Append('-');
689         sb.Append('>');
690         sb.Append(link.Target);
691     }
692 }
693 }

```

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

```

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

```

```

31     /// </summary>
32     protected readonly TLink Break;
33     /// <summary>
34     /// <para>
35     /// The continue.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     protected readonly TLink Continue;
40     /// <summary>
41     /// <para>
42     /// The links.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     protected readonly byte* Links;
47     /// <summary>
48     /// <para>
49     /// The header.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     protected readonly byte* Header;
54
55     /// <summary>
56     /// <para>
57     /// Initializes a new <see cref="LinksRecursionlessSizeBalancedTreeMethodsBase"/>
58     /// instance.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="constants">
63     /// <para>A constants.</para>
64     /// </param>
65     /// <param name="links">
66     /// <para>A links.</para>
67     /// </param>
68     /// <param name="header">
69     /// <para>A header.</para>
70     /// </param>
71     /// </param>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
74     ↪ 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     /// </returns>
91     [MethodImpl(MethodImplOptions.AggressiveInlining)]
92     protected abstract TLink GetTreeRoot();
93
94     /// <summary>
95     /// <para>
96     /// Gets the base part value using the specified link.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="link">
101    /// <para>The link.</para>
102    /// </param>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>

```

```

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

```

```

182
183 /// <summary>
184 /// <para>
185 /// Gets the link reference using the specified link.
186 /// </para>
187 /// <para></para>
188 /// </summary>
189 /// <param name="link">
190 /// <para>The link.</para>
191 /// <para></para>
192 /// </param>
193 /// <returns>
194 /// <para>A ref raw link of t link</para>
195 /// <para></para>
196 /// </returns>
197 [MethodImpl(MethodImplOptions.AggressiveInlining)]
198 protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
    ↪ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));
199
200 /// <summary>
201 /// <para>
202 /// Gets the link values using the specified link index.
203 /// </para>
204 /// <para></para>
205 /// </summary>
206 /// <param name="linkIndex">
207 /// <para>The link index.</para>
208 /// <para></para>
209 /// </param>
210 /// <returns>
211 /// <para>A list of t link</para>
212 /// <para></para>
213 /// </returns>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
216 {
217     ref var link = ref GetLinkReference(linkIndex);
218     return new Link<TLink>(linkIndex, link.Source, link.Target);
219 }
220
221 /// <summary>
222 /// <para>
223 /// Determines whether this instance first is to the left of second.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <param name="first">
228 /// <para>The first.</para>
229 /// <para></para>
230 /// </param>
231 /// <param name="second">
232 /// <para>The second.</para>
233 /// <para></para>
234 /// </param>
235 /// <returns>
236 /// <para>The bool</para>
237 /// <para></para>
238 /// </returns>
239 [MethodImpl(MethodImplOptions.AggressiveInlining)]
240 protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
241 {
242     ref var firstLink = ref GetLinkReference(first);
243     ref var secondLink = ref GetLinkReference(second);
244     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
245 }
246
247 /// <summary>
248 /// <para>
249 /// Determines whether this instance first is to the right of second.
250 /// </para>
251 /// <para></para>
252 /// </summary>
253 /// <param name="first">
254 /// <para>The first.</para>
255 /// <para></para>
256 /// </param>

```



```

257     /// <param name="second">
258     /// <para>The second.</para>
259     /// <para></para>
260     /// </param>
261     /// <returns>
262     /// <para>The bool</para>
263     /// <para></para>
264     /// </returns>
265     [MethodImpl(MethodImplOptions.AggressiveInlining)]
266     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
267     {
268         ref var firstLink = ref GetLinkReference(first);
269         ref var secondLink = ref GetLinkReference(second);
270         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
271             ↪ secondLink.Source, secondLink.Target);
272     }
273     /// <summary>
274     /// <para>
275     /// The zero.
276     /// </para>
277     /// <para></para>
278     /// </summary>
279     public TLink this[TLink index]
280     {
281         [MethodImpl(MethodImplOptions.AggressiveInlining)]
282         get
283         {
284             var root = GetTreeRoot();
285             if (GreaterOrEqualThan(index, GetSize(root)))
286             {
287                 return Zero;
288             }
289             while (!EqualToZero(root))
290             {
291                 var left = GetLeftOrDefault(root);
292                 var leftSize = GetSizeOrZero(left);
293                 if (LessThan(index, leftSize))
294                 {
295                     root = left;
296                     continue;
297                 }
298                 if (AreEqual(index, leftSize))
299                 {
300                     return root;
301                 }
302                 root = GetRightOrDefault(root);
303                 index = Subtract(index, Increment(leftSize));
304             }
305             return Zero; // TODO: Impossible situation exception (only if tree structure
306                 ↪ broken)
307         }
308     }
309     /// <summary>
310     /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
311     ↪ (концом).
312     /// </summary>
313     /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
314     /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
315     /// <returns>Индекс искомой связи.</returns>
316     [MethodImpl(MethodImplOptions.AggressiveInlining)]
317     public TLink Search(TLink source, TLink target)
318     {
319         var root = GetTreeRoot();
320         while (!EqualToZero(root))
321         {
322             ref var rootLink = ref GetLinkReference(root);
323             var rootSource = rootLink.Source;
324             var rootTarget = rootLink.Target;
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
336             {
337                 return root;
338             }
339         }
340     }

```

```

330         root = GetRightOrDefault(root);
331     }
332     else // node.Key == root.Key
333     {
334         return root;
335     }
336 }
337 return Zero;
338 }
339
340 // TODO: Return indices range instead of references count
341 /// <summary>
342 /// <para>
343 /// Counts the usages using the specified link.
344 /// </para>
345 /// <para></para>
346 /// </summary>
347 /// <param name="link">
348 /// <para>The link.</para>
349 /// <para></para>
350 /// </param>
351 /// <returns>
352 /// <para>The link</para>
353 /// <para></para>
354 /// </returns>
355 [MethodImpl(MethodImplOptions.AggressiveInlining)]
356 public TLink CountUsages(TLink link)
357 {
358     var root = GetTreeRoot();
359     var total = GetSize(root);
360     var totalRightIgnore = Zero;
361     while (!EqualToZero(root))
362     {
363         var @base = GetBasePartValue(root);
364         if (LessOrEqualThan(@base, link))
365         {
366             root = GetRightOrDefault(root);
367         }
368         else
369         {
370             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
371             root = GetLeftOrDefault(root);
372         }
373     }
374     root = GetTreeRoot();
375     var totalLeftIgnore = Zero;
376     while (!EqualToZero(root))
377     {
378         var @base = GetBasePartValue(root);
379         if (GreaterOrEqualThan(@base, link))
380         {
381             root = GetLeftOrDefault(root);
382         }
383         else
384         {
385             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
386             root = GetRightOrDefault(root);
387         }
388     }
389     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
390 }
391
392 /// <summary>
393 /// <para>
394 /// Eaches the usage using the specified base.
395 /// </para>
396 /// <para></para>
397 /// </summary>
398 /// <param name="@base">
399 /// <para>The base.</para>
400 /// <para></para>
401 /// </param>
402 /// <param name="handler">
403 /// <para>The handler.</para>
404 /// <para></para>
405 /// </param>
406 /// <returns>
407 /// <para>The link</para>

```

```

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

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Delegates;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.United.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the links size balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizeBalancedTreeMethods{TLink}"/>
21     /// <seealso cref="ILinksTreeMethods{TLink}"/>
22     public unsafe abstract class LinksSizeBalancedTreeMethodsBase<TLink> :
23     ↪ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
26         ↪ UncheckedConverter<TLink, long>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The break.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         protected readonly TLink Break;
35
36         /// <summary>
37         /// <para>
38         /// The continue.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         protected readonly TLink Continue;
43
44         /// <summary>
45         /// <para>
46         /// The links.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly byte* Links;
51
52         /// <summary>
53         /// <para>
54         /// The header.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         protected readonly byte* Header;
59
60         /// <summary>
61         /// <para>
62         /// Initializes a new <see cref="LinksSizeBalancedTreeMethodsBase"/> instance.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <param name="constants">
67         /// <para>A constants.</para>
68         /// <para></para>
69         /// </param>
70         /// <param name="links">
71         /// <para>A links.</para>
72         /// <para></para>
73         /// </param>
74         /// <param name="header">
75         /// <para>A header.</para>
76         /// <para></para>
77         /// </param>
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         protected LinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
80         ↪ byte* header)
81         {

```

```

76     Links = links;
77     Header = header;
78     Break = constants.Break;
79     Continue = constants.Continue;
80 }
81
82 /// <summary>
83 /// <para>
84 /// Gets the tree root.
85 /// </para>
86 /// <para></para>
87 /// </summary>
88 /// <returns>
89 /// <para>The link</para>
90 /// <para></para>
91 /// </returns>
92 [MethodImpl(MethodImplOptions.AggressiveInlining)]
93 protected abstract TLink GetTreeRoot();
94
95 /// <summary>
96 /// <para>
97 /// Gets the base part value using the specified link.
98 /// </para>
99 /// <para></para>
100 /// </summary>
101 /// <param name="link">
102 /// <para>The link.</para>
103 /// <para></para>
104 /// </param>
105 /// <returns>
106 /// <para>The link</para>
107 /// <para></para>
108 /// </returns>
109 [MethodImpl(MethodImplOptions.AggressiveInlining)]
110 protected abstract TLink GetBasePartValue(TLink link);
111
112 /// <summary>
113 /// <para>
114 /// Determines whether this instance first is to the right of second.
115 /// </para>
116 /// <para></para>
117 /// </summary>
118 /// <param name="source">
119 /// <para>The source.</para>
120 /// <para></para>
121 /// </param>
122 /// <param name="target">
123 /// <para>The target.</para>
124 /// <para></para>
125 /// </param>
126 /// <param name="rootSource">
127 /// <para>The root source.</para>
128 /// <para></para>
129 /// </param>
130 /// <param name="rootTarget">
131 /// <para>The root target.</para>
132 /// <para></para>
133 /// </param>
134 /// <returns>
135 /// <para>The bool</para>
136 /// <para></para>
137 /// </returns>
138 [MethodImpl(MethodImplOptions.AggressiveInlining)]
139 protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);
140
141 /// <summary>
142 /// <para>
143 /// Determines whether this instance first is to the left of second.
144 /// </para>
145 /// <para></para>
146 /// </summary>
147 /// <param name="source">
148 /// <para>The source.</para>
149 /// <para></para>
150 /// </param>
151 /// <param name="target">
152 /// <para>The target.</para>

```

```

153     /// <para></para>
154     /// </param>
155     /// <param name="rootSource">
156     /// <para>The root source.</para>
157     /// <para></para>
158     /// </param>
159     /// <param name="rootTarget">
160     /// <para>The root target.</para>
161     /// <para></para>
162     /// </param>
163     /// <returns>
164     /// <para>The bool</para>
165     /// <para></para>
166     /// </returns>
167     [MethodImpl(MethodImplOptions.AggressiveInlining)]
168     protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
        ↪ rootSource, TLink rootTarget);
169
170     /// <summary>
171     /// <para>
172     /// Gets the header reference.
173     /// </para>
174     /// <para></para>
175     /// </summary>
176     /// <returns>
177     /// <para>A ref links header of t link</para>
178     /// <para></para>
179     /// </returns>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪ AsRef<LinksHeader<TLink>>(Header);
182
183     /// <summary>
184     /// <para>
185     /// Gets the link reference using the specified link.
186     /// </para>
187     /// <para></para>
188     /// </summary>
189     /// <param name="link">
190     /// <para>The link.</para>
191     /// <para></para>
192     /// </param>
193     /// <returns>
194     /// <para>A ref raw link of t link</para>
195     /// <para></para>
196     /// </returns>
197     [MethodImpl(MethodImplOptions.AggressiveInlining)]
198     protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
        ↪ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
        ↪ _addressToInt64Converter.Convert(link)));
199
200     /// <summary>
201     /// <para>
202     /// Gets the link values using the specified link index.
203     /// </para>
204     /// <para></para>
205     /// </summary>
206     /// <param name="linkIndex">
207     /// <para>The link index.</para>
208     /// <para></para>
209     /// </param>
210     /// <returns>
211     /// <para>A list of t link</para>
212     /// <para></para>
213     /// </returns>
214     [MethodImpl(MethodImplOptions.AggressiveInlining)]
215     protected virtual IList<TLink>? GetLinkValues(TLink linkIndex)
216     {
217         ref var link = ref GetLinkReference(linkIndex);
218         return new Link<TLink>(linkIndex, link.Source, link.Target);
219     }
220
221     /// <summary>
222     /// <para>
223     /// Determines whether this instance first is to the left of second.
224     /// </para>
225     /// <para></para>
226     /// </summary>

```

```

227    /// <param name="first">
228    /// <para>The first.</para>
229    /// <para></para>
230    /// </param>
231    /// <param name="second">
232    /// <para>The second.</para>
233    /// <para></para>
234    /// </param>
235    /// <returns>
236    /// <para>The bool</para>
237    /// <para></para>
238    /// </returns>
239    [MethodImpl(MethodImplOptions.AggressiveInlining)]
240    protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
241    {
242        ref var firstLink = ref GetLinkReference(first);
243        ref var secondLink = ref GetLinkReference(second);
244        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
245            ↪ secondLink.Source, secondLink.Target);
246    }
247
248    /// <summary>
249    /// <para>
250    /// Determines whether this instance first is to the right of second.
251    /// </para>
252    /// <para></para>
253    /// </summary>
254    /// <param name="first">
255    /// <para>The first.</para>
256    /// <para></para>
257    /// </param>
258    /// <param name="second">
259    /// <para>The second.</para>
260    /// <para></para>
261    /// </param>
262    /// <returns>
263    /// <para>The bool</para>
264    /// <para></para>
265    /// </returns>
266    [MethodImpl(MethodImplOptions.AggressiveInlining)]
267    protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
268    {
269        ref var firstLink = ref GetLinkReference(first);
270        ref var secondLink = ref GetLinkReference(second);
271        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
272            ↪ secondLink.Source, secondLink.Target);
273    }
274
275    /// <summary>
276    /// <para>
277    /// The zero.
278    /// </para>
279    /// <para></para>
280    /// </summary>
281    public TLink this[TLink index]
282    {
283        [MethodImpl(MethodImplOptions.AggressiveInlining)]
284        get
285        {
286            var root = GetTreeRoot();
287            if (GreaterOrEqualThan(index, GetSize(root)))
288            {
289                return Zero;
290            }
291            while (!EqualToZero(root))
292            {
293                var left = GetLeftOrDefault(root);
294                var leftSize = GetSizeOrZero(left);
295                if (LessThan(index, leftSize))
296                {
297                    root = left;
298                    continue;
299                }
300                if (AreEqual(index, leftSize))
301                {
302                    return root;
303                }
304                root = GetRightOrDefault(root);
305            }
306        }
307    }

```

```

303         index = Subtract(index, Increment(leftSize));
304     }
305     return Zero; // TODO: Impossible situation exception (only if tree structure
    ↪ broken)
306 }
307 }
308
309 /// <summary>
310 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
311 /// </summary>
312 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
313 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
314 /// <returns>Индекс искомой связи.</returns>
315 [MethodImpl(MethodImplOptions.AggressiveInlining)]
316 public TLink Search(TLink source, TLink target)
317 {
318     var root = GetTreeRoot();
319     while (!EqualToZero(root))
320     {
321         ref var rootLink = ref GetLinkReference(root);
322         var rootSource = rootLink.Source;
323         var rootTarget = rootLink.Target;
324         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
            ↪ node.Key < root.Key
325         {
326             root = GetLeftOrDefault(root);
327         }
328         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
            ↪ node.Key > root.Key
329         {
330             root = GetRightOrDefault(root);
331         }
332         else // node.Key == root.Key
333         {
334             return root;
335         }
336     }
337     return Zero;
338 }
339
340 // TODO: Return indices range instead of references count
341 /// <summary>
342 /// <para>
343 /// Counts the usages using the specified link.
344 /// </para>
345 /// <para></para>
346 /// </summary>
347 /// <param name="link">
348 /// <para>The link.</para>
349 /// <para></para>
350 /// </param>
351 /// <returns>
352 /// <para>The link</para>
353 /// <para></para>
354 /// </returns>
355 [MethodImpl(MethodImplOptions.AggressiveInlining)]
356 public TLink CountUsages(TLink link)
357 {
358     var root = GetTreeRoot();
359     var total = GetSize(root);
360     var totalRightIgnore = Zero;
361     while (!EqualToZero(root))
362     {
363         var @base = GetBasePartValue(root);
364         if (LessOrEqualThan(@base, link))
365         {
366             root = GetRightOrDefault(root);
367         }
368         else
369         {
370             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
371             root = GetLeftOrDefault(root);
372         }
373     }
374     root = GetTreeRoot();
375     var totalLeftIgnore = Zero;
376     while (!EqualToZero(root))

```



```

377     {
378         var @base = GetBasePartValue(root);
379         if (GreaterOrEqualThan(@base, link))
380         {
381             root = GetLeftOrDefault(root);
382         }
383         else
384         {
385             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
386             root = GetRightOrDefault(root);
387         }
388     }
389     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
390 }
391
392 /// <summary>
393 /// <para>
394 /// Eaches the usage using the specified base.
395 /// </para>
396 /// <para></para>
397 /// </summary>
398 /// <param name="@base">
399 /// <para>The base.</para>
400 /// <para></para>
401 /// </param>
402 /// <param name="handler">
403 /// <para>The handler.</para>
404 /// <para></para>
405 /// </param>
406 /// <returns>
407 /// <para>The link</para>
408 /// <para></para>
409 /// </returns>
410 [MethodImpl(MethodImplOptions.AggressiveInlining)]
411 public TLink EachUsage(TLink @base, ReadHandler<TLink>? handler) => EachUsageCore(@base,
    ↪ GetTreeRoot(), handler);
412
413 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
    ↪ low-level MSIL stack.
414 [MethodImpl(MethodImplOptions.AggressiveInlining)]
415 private TLink EachUsageCore(TLink @base, TLink link, ReadHandler<TLink>? handler)
416 {
417     var @continue = Continue;
418     if (EqualToZero(link))
419     {
420         return @continue;
421     }
422     var linkBasePart = GetBasePartValue(link);
423     var @break = Break;
424     if (GreaterThan(linkBasePart, @base))
425     {
426         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
427         {
428             return @break;
429         }
430     }
431     else if (LessThan(linkBasePart, @base))
432     {
433         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
434         {
435             return @break;
436         }
437     }
438     else //if (linkBasePart == @base)
439     {
440         if (AreEqual(handler(GetLinkValues(link)), @break))
441         {
442             return @break;
443         }
444         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
445         {
446             return @break;
447         }
448         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
449         {
450             return @break;
451         }
452     }
453 }

```

```

453         return @continue;
454     }
455
456     /// <summary>
457     /// <para>
458     /// Prints the node value using the specified node.
459     /// </para>
460     /// <para></para>
461     /// </summary>
462     /// <param name="node">
463     /// <para>The node.</para>
464     /// <para></para>
465     /// </param>
466     /// <param name="sb">
467     /// <para>The sb.</para>
468     /// <para></para>
469     /// </param>
470     [MethodImpl(MethodImplOptions.AggressiveInlining)]
471     protected override void PrintNodeValue(TLink node, StringBuilder sb)
472     {
473         ref var link = ref GetLinkReference(node);
474         sb.Append(' ');
475         sb.Append(link.Source);
476         sb.Append('-');
477         sb.Append('>');
478         sb.Append(link.Target);
479     }
480 }
481 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links sources avl balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="LinksAvlBalancedTreeMethodsBase{TLink}"/>
14     public unsafe class LinksSourcesAvlBalancedTreeMethods<TLink> :
15         ↳ LinksAvlBalancedTreeMethodsBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksSourcesAvlBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public LinksSourcesAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
37             ↳ byte* header) : base(constants, links, header) { }
38
39         /// <summary>
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>

```

```

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

```

```

121
122     /// <summary>
123     /// <para>
124     /// Sets the right using the specified node.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="node">
129     /// <para>The node.</para>
130     /// <para></para>
131     /// </param>
132     /// <param name="right">
133     /// <para>The right.</para>
134     /// <para></para>
135     /// </param>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected override void SetRight(TLink node, TLink right) =>
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 TLink GetSize(TLink node) =>
156         ↪ GetSizeValue(GetLinkReference(node).SizeAsSource);
157
158     /// <summary>
159     /// <para>
160     /// Sets the size using the specified node.
161     /// </para>
162     /// <para></para>
163     /// </summary>
164     /// <param name="node">
165     /// <para>The node.</para>
166     /// <para></para>
167     /// </param>
168     /// <param name="size">
169     /// <para>The size.</para>
170     /// <para></para>
171     /// </param>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     protected override void SetSize(TLink node, TLink size) => SetSizeValue(ref
174         ↪ GetLinkReference(node).SizeAsSource, size);
175
176     /// <summary>
177     /// <para>
178     /// Determines whether this instance get left is child.
179     /// </para>
180     /// <para></para>
181     /// </summary>
182     /// <param name="node">
183     /// <para>The node.</para>
184     /// <para></para>
185     /// </param>
186     /// <returns>
187     /// <para>The bool</para>
188     /// <para></para>
189     /// </returns>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override bool GetLeftIsChild(TLink node) =>
192         ↪ GetLeftIsChildValue(GetLinkReference(node).SizeAsSource);
193
194     /// <summary>
195     /// <para>
196     /// Sets the left is child using the specified node.
197     /// </para>
198     /// <para></para>

```

```

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

```

```

269    /// <para>The value.</para>
270    /// <para></para>
271    /// </param>
272    [MethodImpl(MethodImplOptions.AggressiveInlining)]
273    protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
    ↪ GetLinkReference(node).SizeAsSource, value);

274
275    /// <summary>
276    /// <para>
277    /// Gets the tree root.
278    /// </para>
279    /// <para></para>
280    /// </summary>
281    /// <returns>
282    /// <para>The link</para>
283    /// <para></para>
284    /// </returns>
285    [MethodImpl(MethodImplOptions.AggressiveInlining)]
286    protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
287
288    /// <summary>
289    /// <para>
290    /// Gets the base part value using the specified link.
291    /// </para>
292    /// <para></para>
293    /// </summary>
294    /// <param name="link">
295    /// <para>The link.</para>
296    /// <para></para>
297    /// </param>
298    /// <returns>
299    /// <para>The link</para>
300    /// <para></para>
301    /// </returns>
302    [MethodImpl(MethodImplOptions.AggressiveInlining)]
303    protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
304
305    /// <summary>
306    /// <para>
307    /// Determines whether this instance first is to the left of second.
308    /// </para>
309    /// <para></para>
310    /// </summary>
311    /// <param name="firstSource">
312    /// <para>The first source.</para>
313    /// <para></para>
314    /// </param>
315    /// <param name="firstTarget">
316    /// <para>The first target.</para>
317    /// <para></para>
318    /// </param>
319    /// <param name="secondSource">
320    /// <para>The second source.</para>
321    /// <para></para>
322    /// </param>
323    /// <param name="secondTarget">
324    /// <para>The second target.</para>
325    /// <para></para>
326    /// </param>
327    /// <returns>
328    /// <para>The bool</para>
329    /// <para></para>
330    /// </returns>
331    [MethodImpl(MethodImplOptions.AggressiveInlining)]
332    protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
    ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
333
334    /// <summary>
335    /// <para>
336    /// Determines whether this instance first is to the right of second.
337    /// </para>
338    /// <para></para>
339    /// </summary>
340    /// <param name="firstSource">
341    /// <para>The first source.</para>
342    /// <para></para>
343    /// </param>

```

```

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

```

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

```

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

```

```

34 [MethodImpl(MethodImplOptions.AggressiveInlining)]
35 public LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
    ↳ byte* links, byte* header) : base(constants, links, header) { }
36
37 /// <summary>
38 /// <para>
39 /// Gets the left reference using the specified node.
40 /// </para>
41 /// <para></para>
42 /// </summary>
43 /// <param name="node">
44 /// <para>The node.</para>
45 /// <para></para>
46 /// </param>
47 /// <returns>
48 /// <para>The ref link</para>
49 /// <para></para>
50 /// </returns>
51 [MethodImpl(MethodImplOptions.AggressiveInlining)]
52 protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ 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 TLink GetRightReference(TLink 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 TLink GetLeft(TLink 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 TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
104
105 /// <summary>
106 /// <para>
107 /// Sets the left using the specified node.
108 /// </para>

```



```

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

```

```

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

```

```

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

```

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

```

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

```

```

51 [MethodImpl(MethodImplOptions.AggressiveInlining)]
52 protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ 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 TLink GetRightReference(TLink 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 TLink GetLeft(TLink 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 TLink GetRight(TLink node) => GetLinkReference(node).RightAsSource;
104
105 /// <summary>
106 /// <para>
107 /// Sets the left using the specified node.
108 /// </para>
109 /// <para></para>
110 /// </summary>
111 /// <param name="node">
112 /// <para>The node.</para>
113 /// <para></para>
114 /// </param>
115 /// <param name="left">
116 /// <para>The left.</para>
117 /// <para></para>
118 /// </param>
119 [MethodImpl(MethodImplOptions.AggressiveInlining)]
120 protected override void SetLeft(TLink node, TLink left) =>
    ↳ GetLinkReference(node).LeftAsSource = left;
121
122 /// <summary>
123 /// <para>
124 /// Sets the right using the specified node.
125 /// </para>

```

```

126     /// <para></para>
127     /// </summary>
128     /// <param name="node">
129     /// <para>The node.</para>
130     /// <para></para>
131     /// </param>
132     /// <param name="right">
133     /// <para>The right.</para>
134     /// <para></para>
135     /// </param>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected override void SetRight(TLink node, TLink right) =>
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 TLink GetSize(TLink node) => 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(TLink node, TLink size) =>
173         ↪ GetLinkReference(node).SizeAsSource = size;
174
175     /// <summary>
176     /// <para>
177     /// Gets the tree root.
178     /// </para>
179     /// <para></para>
180     /// </summary>
181     /// <returns>
182     /// <para>The link</para>
183     /// <para></para>
184     /// </returns>
185     [MethodImpl(MethodImplOptions.AggressiveInlining)]
186     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
187
188     /// <summary>
189     /// <para>
190     /// Gets the base part value using the specified link.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     /// <param name="link">
195     /// <para>The link.</para>
196     /// <para></para>
197     /// </param>
198     /// <returns>
199     /// <para>The link</para>
200     /// <para></para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;

```

```

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(TLink firstSource, TLink firstTarget,
230     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
231     ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
232     /// <summary>
233     /// <para>
234     /// Determines whether this instance first is to the right of second.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="firstTarget">
243     /// <para>The first target.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
260     ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
261     ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
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(TLink node)
274     {
275         ref var link = ref GetLinkReference(node);

```

```

275         link.LeftAsSource = Zero;
276         link.RightAsSource = Zero;
277         link.SizeAsSource = Zero;
278     }
279 }
280 }

```

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

```

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

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

```



```

143     /// <para></para>
144     /// </summary>
145     /// <param name="node">
146     /// <para>The node.</para>
147     /// <para></para>
148     /// </param>
149     /// <returns>
150     /// <para>The link</para>
151     /// <para></para>
152     /// </returns>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     protected override TLink GetSize(TLink node) =>
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(TLink node, TLink size) => SetSizeValue(ref
173         ↪ 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(TLink 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(TLink 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>

```

```

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

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the links targets recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14     public unsafe class LinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
15         ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
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<TLink> constants,
38             ↳ 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 TLink GetLeftReference(TLink node) => ref
56             ↳ GetLinkReference(node).LeftAsTarget;
57     }
58 }

```

```

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

```

```

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

```

```

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

```

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

```

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

```



```

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

```

```

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

```

```

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

231
232     /// <summary>
233     /// <para>
234     /// Determines whether this instance first is to the right of second.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="firstTarget">
243     /// <para>The first target.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
    ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));

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

```

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

```

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

```

```

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

```

```

87     /// <para>A memory reservation step.</para>
88     /// <para></para>
89     /// </param>
90     /// <param name="constants">
91     /// <para>A constants.</para>
92     /// <para></para>
93     /// </param>
94     /// <param name="indexTreeType">
95     /// <para>A index tree type.</para>
96     /// <para></para>
97     /// </param>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep,
    ↪ LinksConstants<TLink> constants, IndexTreeType indexTreeType) : base(memory,
    ↪ memoryReservationStep, constants)
100     {
101         if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
102         {
103             _createSourceTreeMethods = () => new
104                 ↪ LinksSourcesAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
105             _createTargetTreeMethods = () => new
106                 ↪ LinksTargetsAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
107         }
108         else
109         {
110             _createSourceTreeMethods = () => new
111                 ↪ LinksSourcesSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
112             _createTargetTreeMethods = () => new
113                 ↪ LinksTargetsSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
114         }
115         Init(memory, memoryReservationStep);
116     }
117
118     /// <summary>
119     /// <para>
120     /// Sets the pointers using the specified memory.
121     /// </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         _links = (byte*)memory.Pointer;
131         _header = _links;
132         SourcesTreeMethods = _createSourceTreeMethods();
133         TargetsTreeMethods = _createTargetTreeMethods();
134         UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_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 t link</para>
159     /// <para></para>
160     /// </returns>
161     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

159     protected override ref LinkHeader<TLink> GetHeaderReference() => ref
160         ↪ AsRef<LinkHeader<TLink>>(_header);
161
162     /// <summary>
163     /// <para>
164     /// Gets the link reference using the specified link index.
165     /// </para>
166     /// </summary>
167     /// <param name="linkIndex">
168     /// <para>The link index.</para>
169     /// </param>
170     /// </returns>
171     /// <para>A ref raw link of t link</para>
172     /// </returns>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override ref RawLink<TLink> GetLinkReference(TLink linkIndex) => ref
175         ↪ AsRef<RawLink<TLink>>(_links + (LinkSizeInBytes * Convert.ToInt64(linkIndex)));
176
177 }
178 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Disposables;
5  using Platform.Singletons;
6  using Platform.Converters;
7  using Platform.Numbers;
8  using Platform.Memory;
9  using Platform.Data.Exceptions;
10 using Platform.Delegates;
11
12 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
13
14 namespace Platform.Data.Doublets.Memory.United.Generic
15 {
16     /// <summary>
17     /// <para>
18     /// Represents the united memory links base.
19     /// </para>
20     /// </summary>
21     /// <seealso cref="DisposableBase"/>
22     /// <seealso cref="ILinks{TLink}"/>
23     public abstract class UnitedMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink> where
24         ↪ TLink : struct
25     {
26         private static readonly EqualityComparer<TLink> _equalityComparer =
27             ↪ EqualityComparer<TLink>.Default;
28         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
29         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
30             ↪ UncheckedConverter<TLink, long>.Default;
31         private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
32             ↪ UncheckedConverter<long, TLink>.Default;
33         private static readonly TLink _zero = default;
34         private static readonly TLink _one = Arithmetic.Increment(_zero);
35
36         /// <summary>Возвращает размер одной связи в байтах.</summary>
37         /// <remarks>
38         /// Используется только во вне класса, не рекомендуется использовать внутри.
39         /// Так как во вне не обязательно будет доступен unsafe C#.
40         /// </remarks>
41         public static readonly long LinkSizeInBytes = RawLink<TLink>.SizeInBytes;
42
43         /// <summary>
44         /// <para>
45         /// The size in bytes.
46         /// </para>
47         /// </summary>
48         public static readonly long LinkHeaderSizeInBytes = LinkHeader<TLink>.SizeInBytes;
49
50         /// <summary>
51         /// <para>
52         /// The link size in bytes.
53         /// </para>
54         /// </summary>
55     }
56 }

```

```

53     /// </summary>
54     public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
55
56     /// <summary>
57     /// <para>
58     /// The memory.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     protected readonly IResizableDirectMemory _memory;
63     /// <summary>
64     /// <para>
65     /// The memory reservation step.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     protected readonly long _memoryReservationStep;
70
71     /// <summary>
72     /// <para>
73     /// The targets tree methods.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     protected ILinksTreeMethods<TLink> TargetsTreeMethods;
78     /// <summary>
79     /// <para>
80     /// The sources tree methods.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     protected ILinksTreeMethods<TLink> SourcesTreeMethods;
85     // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
86     // → нужно использовать не список а дерево, так как так можно быстрее проверить на
87     // → наличие связи внутри
88     /// <summary>
89     /// <para>
90     /// The unused links list methods.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     protected ILinksListMethods<TLink> UnusedLinksListMethods;
95
96     /// <summary>
97     /// Возвращает общее число связей находящихся в хранилище.
98     /// </summary>
99     protected virtual TLink Total
100     {
101         [MethodImpl(MethodImplOptions.AggressiveInlining)]
102         get
103         {
104             ref var header = ref GetHeaderReference();
105             return Subtract(header.AllocatedLinks, header.FreeLinks);
106         }
107     }
108
109     /// <summary>
110     /// <para>
111     /// Gets the constants value.
112     /// </para>
113     /// <para></para>
114     /// </summary>
115     public virtual LinksConstants<TLink> Constants
116     {
117         [MethodImpl(MethodImplOptions.AggressiveInlining)]
118         get;
119     }
120
121     /// <summary>
122     /// <para>
123     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
124     /// </para>
125     /// <para></para>
126     /// </summary>
127     /// <param name="memory">
128     /// <para>A memory.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="memoryReservationStep">

```

```

130 /// <para>A memory reservation step.</para>
131 /// <para></para>
132 /// </param>
133 /// <param name="constants">
134 /// <para>A constants.</para>
135 /// <para></para>
136 /// </param>
137 [MethodImpl(MethodImplOptions.AggressiveInlining)]
138 protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
    ↳ memoryReservationStep, LinksConstants<TLink> constants)
139 {
140     _memory = memory;
141     memoryReservationStep = memoryReservationStep;
142     Constants = constants;
143 }
144
145 /// <summary>
146 /// <para>
147 /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <param name="memory">
152 /// <para>A memory.</para>
153 /// <para></para>
154 /// </param>
155 /// <param name="memoryReservationStep">
156 /// <para>A memory reservation step.</para>
157 /// <para></para>
158 /// </param>
159 [MethodImpl(MethodImplOptions.AggressiveInlining)]
160 protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
    ↳ memoryReservationStep) : this(memory, memoryReservationStep,
    ↳ Default<LinksConstants<TLink>>.Instance) { }
161
162 /// <summary>
163 /// <para>
164 /// Inits the memory.
165 /// </para>
166 /// <para></para>
167 /// </summary>
168 /// <param name="memory">
169 /// <para>The memory.</para>
170 /// <para></para>
171 /// </param>
172 /// <param name="memoryReservationStep">
173 /// <para>The memory reservation step.</para>
174 /// <para></para>
175 /// </param>
176 [MethodImpl(MethodImplOptions.AggressiveInlining)]
177 protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
178 {
179     if (memory.ReservedCapacity < memoryReservationStep)
180     {
181         memory.ReservedCapacity = memoryReservationStep;
182     }
183     SetPointers(memory);
184     ref var header = ref GetHeaderReference();
185     // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
186     memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
    ↳ LinkHeaderSizeInBytes;
187     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
188     header.ReservedLinks = ConvertToAddress((memory.ReservedCapacity -
    ↳ LinkHeaderSizeInBytes) / LinkSizeInBytes);
189 }
190
191 /// <summary>
192 /// <para>
193 /// Counts the substitution.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <param name="restriction">
198 /// <para>The substitution.</para>
199 /// <para></para>
200 /// </param>
201 /// <exception cref="NotSupportedException">
202 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>

```



```

203 /// <para></para>
204 /// </exception>
205 /// <returns>
206 /// <para>The link</para>
207 /// <para></para>
208 /// </returns>
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 public virtual TLink Count(IList<TLink>? restriction)
211 {
212     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
213     if (restriction.Count == 0)
214     {
215         return Total;
216     }
217     var constants = Constants;
218     var any = constants.Any;
219     var index = restriction[constants.IndexPart];
220     if (restriction.Count == 1)
221     {
222         if (AreEqual(index, any))
223         {
224             return Total;
225         }
226         return Exists(index) ? GetOne() : GetZero();
227     }
228     if (restriction.Count == 2)
229     {
230         var value = restriction[1];
231         if (AreEqual(index, any))
232         {
233             if (AreEqual(value, any))
234             {
235                 return Total; // Any - как отсутствие ограничения
236             }
237             return Add(SourcesTreeMethods.CountUsages(value),
238                 ↪ TargetsTreeMethods.CountUsages(value));
239         }
240         else
241         {
242             if (!Exists(index))
243             {
244                 return GetZero();
245             }
246             if (AreEqual(value, any))
247             {
248                 return GetOne();
249             }
250             ref var storedLinkValue = ref GetLinkReference(index);
251             if (AreEqual(storedLinkValue.Source, value) ||
252                 ↪ AreEqual(storedLinkValue.Target, value))
253             {
254                 return GetOne();
255             }
256             return GetZero();
257         }
258     }
259     if (restriction.Count == 3)
260     {
261         var source = restriction[constants.SourcePart];
262         var target = restriction[constants.TargetPart];
263         if (AreEqual(index, any))
264         {
265             if (AreEqual(source, any) && AreEqual(target, any))
266             {
267                 return Total;
268             }
269             else if (AreEqual(source, any))
270             {
271                 return TargetsTreeMethods.CountUsages(target);
272             }
273             else if (AreEqual(target, any))
274             {
275                 return SourcesTreeMethods.CountUsages(source);
276             }
277             else //if(source != Any && target != Any)
278             {
279                 // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
280                 var link = SourcesTreeMethods.Search(source, target);

```

```

279         return AreEqual(link, constants.Null) ? GetZero() : GetOne();
280     }
281 }
282 else
283 {
284     if (!Exists(index))
285     {
286         return GetZero();
287     }
288     if (AreEqual(source, any) && AreEqual(target, any))
289     {
290         return GetOne();
291     }
292     ref var storedLinkValue = ref GetLinkReference(index);
293     if (!AreEqual(source, any) && !AreEqual(target, any))
294     {
295         if (AreEqual(storedLinkValue.Source, source) &&
296             ↪ AreEqual(storedLinkValue.Target, target))
297         {
298             return GetOne();
299         }
300         return GetZero();
301     }
302     var value = default(TLink);
303     if (AreEqual(source, any))
304     {
305         value = target;
306     }
307     if (AreEqual(target, any))
308     {
309         value = source;
310     }
311     if (AreEqual(storedLinkValue.Source, value) ||
312         ↪ AreEqual(storedLinkValue.Target, value))
313     {
314         return GetOne();
315     }
316     return GetZero();
317 }
318 }
319 }
320 throw new NotSupportedException("Другие размеры и способы ограничений не
321 ↪ поддерживаются.");
322 }
323
324 /// <summary>
325 /// <para>
326 /// Eaches the handler.
327 /// </para>
328 /// <para></para>
329 /// </summary>
330 /// <param name="handler">
331 /// <para>The handler.</para>
332 /// <para></para>
333 /// </param>
334 /// <param name="restriction">
335 /// <para>The substitution.</para>
336 /// <para></para>
337 /// </param>
338 /// <exception cref="NotSupportedException">
339 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
340 /// <para></para>
341 /// </exception>
342 /// <returns>
343 /// <para>The link</para>
344 /// <para></para>
345 /// </returns>
346 [MethodImpl(MethodImplOptions.AggressiveInlining)]
347 public virtual TLink Each(ICollection<TLink>? restriction, ReadHandler<TLink>? handler)
348 {
349     var constants = Constants;
350     var @break = constants.Break;
351     if (restriction.Count == 0)
352     {
353         for (var link = GetOne(); LessOrEqualThan(link,
354             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
355         {
356             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
357             {

```

```

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<TLink>(), 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<TLink>(), handler);
381         }
382         if (AreEqual(Each(new Link<TLink>(index, value, any), handler), @break))
383         {
384             return @break;
385         }
386         return Each(new Link<TLink>(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<TLink>(), 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     else
432     {
433         if (!Exists(index))
434         {
435             return @continue;
436         }
437         if (AreEqual(source, any) && AreEqual(target, any))
438         {
439             return handler(GetLinkStruct(index));
440         }
441         ref var storedLinkValue = ref GetLinkReference(index);
442         if (!AreEqual(source, any) && !AreEqual(target, any))
443         {
444             if (AreEqual(storedLinkValue.Source, source) &&
445                 AreEqual(storedLinkValue.Target, target))
446             {
447                 return handler(GetLinkStruct(index));
448             }
449             return @continue;
450         }
451         var value = default(TLink);
452         if (AreEqual(source, any))
453         {
454             value = target;
455         }
456         if (AreEqual(target, any))
457         {
458             value = source;
459         }
460         if (AreEqual(storedLinkValue.Source, value) ||
461             AreEqual(storedLinkValue.Target, value))
462         {
463             return handler(GetLinkStruct(index));
464         }
465         return @continue;
466     }
467 }
468 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↳ поддерживаются.");
469 }
470
471 /// <remarks>
472 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
473 ↳ в другом месте (но не в менеджере памяти, а в логике Links)
474 /// </remarks>
475 [MethodImpl(MethodImplOptions.AggressiveInlining)]
476 public virtual TLink Update(IList<TLink>? restriction, IList<TLink>? substitution,
477     ↳ WriteHandler<TLink>? handler)
478 {
479     var constants = Constants;
480     var @null = constants.Null;
481     var linkIndex = restriction[constants.IndexPart];
482     var before = GetLinkStruct(linkIndex);
483     ref var link = ref GetLinkReference(linkIndex);
484     ref var header = ref GetHeaderReference();
485     ref var firstAsSource = ref header.RootAsSource;
486     ref var firstAsTarget = ref header.RootAsTarget;
487     // Будет корректно работать только в том случае, если пространство выделенной связи
488     ↳ предварительно заполнено нулями
489     if (!AreEqual(link.Source, @null))
490     {
491         SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
492     }
493     if (!AreEqual(link.Target, @null))
494     {
495         TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
496     }
497     link.Source = substitution[constants.SourcePart];
498     link.Target = substitution[constants.TargetPart];
499     if (!AreEqual(link.Source, @null))
500     {
501         SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
502     }
503     if (!AreEqual(link.Target, @null))
504     {
505         TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
506     }
507 }

```

```

504     return handler?.Invoke(before, GetLinkStruct(linkIndex)) ?? Constants.Continue;
505 }
506
507 /// <remarks>
508 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
509 /// </remarks>
510 [MethodImpl(MethodImplOptions.AggressiveInlining)]
511 public virtual TLink Create(IList<TLink>? substitution, WriteHandler<TLink>? handler)
512 {
513     ref var header = ref GetHeaderReference();
514     var freeLink = header.FirstFreeLink;
515     if (!AreEqual(freeLink, Constants.Null))
516     {
517         UnusedLinksListMethods.Detach(freeLink);
518     }
519     else
520     {
521         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
522         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
523         {
524             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
525         }
526         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
527         {
528             _memory.ReservedCapacity += _memory.ReservationStep;
529             SetPointers(_memory);
530             header = ref GetHeaderReference();
531             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
532                 ↪ LinkSizeInBytes);
533         }
534         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
535         _memory.UsedCapacity += LinkSizeInBytes;
536     }
537     return handler?.Invoke(null, new Link<TLink>(freeLink, Constants.Null,
538         ↪ Constants.Null)) ?? Constants.Continue;
539 }
540
541 /// <summary>
542 /// <para>
543 /// Deletes the substitution.
544 /// </para>
545 /// <para></para>
546 /// </summary>
547 /// <param name="restriction">
548 /// <para>The substitution.</para>
549 /// <para></para>
550 /// </param>
551 [MethodImpl(MethodImplOptions.AggressiveInlining)]
552 public virtual TLink Delete(IList<TLink>? restriction, WriteHandler<TLink>? handler)
553 {
554     ref var header = ref GetHeaderReference();
555     var link = restriction[Constants.IndexPart];
556     var before = GetLinkStruct(link);
557     if (LessThan(link, header.AllocatedLinks))
558     {
559         UnusedLinksListMethods.AttachAsFirst(link);
560         return handler?.Invoke(before, null) ?? Constants.Continue;
561     }
562     else if (AreEqual(link, header.AllocatedLinks))
563     {
564         header.AllocatedLinks = Decrement(header.AllocatedLinks);
565         _memory.UsedCapacity -= LinkSizeInBytes;
566         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
567         // пока не дойдём до первой существующей связи
568         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
569         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
570             ↪ IsUnusedLink(header.AllocatedLinks))
571         {
572             UnusedLinksListMethods.Detach(header.AllocatedLinks);
573             header.AllocatedLinks = Decrement(header.AllocatedLinks);
574             _memory.UsedCapacity -= LinkSizeInBytes;
575         }
576         return handler?.Invoke(before, null) ?? Constants.Continue;
577     }
578     return Constants.Continue;
579 }

```

```

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

```

```

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

```

```

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

```



```

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

```

```

880    /// <para>The link</para>
881    /// <para></para>
882    /// </returns>
883    [MethodImpl(MethodImplOptions.AggressiveInlining)]
884    protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
    ↪    second);

885
886    /// <summary>
887    /// <para>
888    /// Subtracts the first.
889    /// </para>
890    /// <para></para>
891    /// </summary>
892    /// <param name="first">
893    /// <para>The first.</para>
894    /// <para></para>
895    /// </param>
896    /// <param name="second">
897    /// <para>The second.</para>
898    /// <para></para>
899    /// </param>
900    /// <returns>
901    /// <para>The link</para>
902    /// <para></para>
903    /// </returns>
904    [MethodImpl(MethodImplOptions.AggressiveInlining)]
905    protected virtual TLink Subtract(TLink first, TLink second) =>
    ↪    Arithmetic<TLink>.Subtract(first, second);

906
907    /// <summary>
908    /// <para>
909    /// Increments the link.
910    /// </para>
911    /// <para></para>
912    /// </summary>
913    /// <param name="link">
914    /// <para>The link.</para>
915    /// <para></para>
916    /// </param>
917    /// <returns>
918    /// <para>The link</para>
919    /// <para></para>
920    /// </returns>
921    [MethodImpl(MethodImplOptions.AggressiveInlining)]
922    protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
923
924    /// <summary>
925    /// <para>
926    /// Decrements the link.
927    /// </para>
928    /// <para></para>
929    /// </summary>
930    /// <param name="link">
931    /// <para>The link.</para>
932    /// <para></para>
933    /// </param>
934    /// <returns>
935    /// <para>The link</para>
936    /// <para></para>
937    /// </returns>
938    [MethodImpl(MethodImplOptions.AggressiveInlining)]
939    protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.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{TLink}" />
17     /// <seealso cref="ILinksListMethods{TLink}" />
18     public unsafe class UnusedLinksListMethods<TLink> :
19         ↳ AbsoluteCircularDoublyLinkedListMethods<TLink>, ILinksListMethods<TLink>
20     {
21         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
22             ↳ UncheckedConverter<TLink, long>.Default;
23         private readonly byte* _links;
24         private readonly byte* _header;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UnusedLinksListMethods" /> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="links">
33         /// <para>A links.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UnusedLinksListMethods(byte* links, byte* header)
42         {
43             _links = links;
44             _header = header;
45         }
46
47         /// <summary>
48         /// <para>
49         /// Gets the header reference.
50         /// </para>
51         /// <para></para>

```

```

50     /// </summary>
51     /// <returns>
52     /// <para>A ref links header of t link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
57         ↳ AsRef<LinksHeader<TLink>>(_header);
58
59     /// <summary>
60     /// <para>
61     /// Gets the link 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<TLink> GetLinkReference(TLink link) => ref
75         ↳ AsRef<RawLink<TLink>>(_links + (RawLink<TLink>.SizeInBytes *
76         ↳ _addressToInt64Converter.Convert(link)));
77
78     /// <summary>
79     /// <para>
80     /// Gets the first.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <returns>
85     /// <para>The link</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
90
91     /// <summary>
92     /// <para>
93     /// Gets the last.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <returns>
98     /// <para>The link</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
103
104    /// <summary>
105    /// <para>
106    /// Gets the previous using the specified element.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="element">
111    /// <para>The element.</para>
112    /// <para></para>
113    /// </param>
114    /// <returns>
115    /// <para>The link</para>
116    /// <para></para>
117    /// </returns>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override TLink GetPrevious(TLink element) => GetLinkReference(element).Source;
120
121    /// <summary>
122    /// <para>
123    /// Gets the next using the specified element.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="element">

```

```

125     /// <para>The element.</para>
126     /// <para></para>
127     /// </param>
128     /// <returns>
129     /// <para>The link</para>
130     /// <para></para>
131     /// </returns>
132     [MethodImpl(MethodImplOptions.AggressiveInlining)]
133     protected override TLink GetNext(TLink element) => GetLinkReference(element).Target;
134
135     /// <summary>
136     /// <para>
137     /// Gets the size.
138     /// </para>
139     /// <para></para>
140     /// </summary>
141     /// <returns>
142     /// <para>The link</para>
143     /// <para></para>
144     /// </returns>
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     protected override TLink GetSize() => GetHeaderReference().FreeLinks;
147
148     /// <summary>
149     /// <para>
150     /// Sets the first using the specified element.
151     /// </para>
152     /// <para></para>
153     /// </summary>
154     /// <param name="element">
155     /// <para>The element.</para>
156     /// <para></para>
157     /// </param>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
        ↪ 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(TLink 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(TLink element, TLink previous) =>
        ↪ 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(TLink element, TLink next) =>
207         ↪ GetLinkReference(element).Target = next;
208
209     /// <summary>
210     /// <para>
211     /// Sets the size using the specified size.
212     /// </para>
213     /// <para></para>
214     /// </summary>
215     /// <param name="size">
216     /// <para>The size.</para>
217     /// <para></para>
218     /// </param>
219     [MethodImpl(MethodImplOptions.AggressiveInlining)]
220     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
221 }

```

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

```

1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.United
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLink<TLink> : IEquatable<RawLink<TLink>>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
19             ↪ EqualityComparer<TLink>.Default;
20
21         /// <summary>
22         /// <para>
23         /// The size.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLink<TLink>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLink Source;
36
37         /// <summary>
38         /// <para>
39         /// The target.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink Target;
44
45         /// <summary>
46         /// <para>
47         /// The left as source.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public TLink LeftAsSource;
52
53         /// <summary>
54         /// <para>
55         /// The right as source.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         public TLink RightAsSource;
60     }
61 }

```

```

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

```

```

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

```

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

```

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

```



```

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

```

```

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

```

```

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

```

```

286     /// </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     /// <summary>

```

```

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

```

#### 1.94 ./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>

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```

```

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

```

```

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

```

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

```
1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links sources size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksSourcesSizeBalancedTreeMethods :
15        ↳ UInt32LinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt32LinksSourcesSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="header">
32        /// <para>A header.</para>
33        /// <para></para>
34        /// </param>
35        public UInt32LinksSourcesSizeBalancedTreeMethods(LinksConstants<uint> constants,
36            ↳ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
37
38        /// <summary>
39        /// <para>
40        /// Gets the left reference using the specified node.
41        /// </para>
42        /// <para></para>
43        /// </summary>
44        /// <param name="node">
45        /// <para>The node.</para>
46        /// <para></para>
47        /// </param>
48        /// <returns>
49        /// <para>The ref uint</para>
50        /// <para></para>
51        /// </returns>
52        [MethodImpl(MethodImplOptions.AggressiveInlining)]
53        protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
54
55        /// <summary>
56        /// <para>
57        /// Gets the right reference using the specified node.
58        /// </para>
59        /// <para></para>
60        /// </summary>
61        /// <param name="node">
62        /// <para>The node.</para>
63        /// <para></para>
64        /// </param>
65        /// <returns>
66        /// <para>The ref uint</para>
67        /// <para></para>
68        /// </returns>
69        [MethodImpl(MethodImplOptions.AggressiveInlining)]
70        protected override ref uint GetRightReference(uint node) => ref
71            ↳ Links[node].RightAsSource;
72
73        /// <summary>
74        /// <para>
75        /// Gets the left using the specified node.
76        /// </para>
77        /// <para></para>
```

```

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

```



```

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

```

```

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

```

## 1.97 ./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>

```

```

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

```

```

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

```

```

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

```

```

245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
        ↪ uint secondSource, uint secondTarget)
        ↪ => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
        ↪ secondSource);
261
262     /// <summary>
263     /// <para>
264     /// Clears the node using the specified node.
265     /// </para>
266     /// <para></para>
267     /// </summary>
268     /// <param name="node">
269     /// <para>The node.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void ClearNode(uint node)
274     {
275         ref var link = ref Links[node];
276         link.LeftAsTarget = 0U;
277         link.RightAsTarget = 0U;
278         link.SizeAsTarget = 0U;
279     }
280 }
281 }

```

## 1.98 ./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
69    ↪ Links[node].RightAsTarget;
70
71    /// <summary>
72    /// <para>
73    /// Gets the left using the specified node.
74    /// </para>
75    /// <para></para>
76    /// </summary>
77    /// <param name="node">
78    /// <para>The node.</para>
79    /// <para></para>
80    /// </param>
81    /// <returns>
82    /// <para>The uint</para>
83    /// <para></para>
84    /// </returns>
85    [MethodImpl(MethodImplOptions.AggressiveInlining)]
86    protected override uint GetLeft(uint node) => Links[node].LeftAsTarget;
87
88    /// <summary>
89    /// <para>
90    /// Gets the right using the specified node.
91    /// </para>
92    /// <para></para>
93    /// </summary>
94    /// <param name="node">
95    /// <para>The node.</para>
96    /// <para></para>
97    /// </param>
98    /// <returns>
99    /// <para>The uint</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override uint GetRight(uint node) => Links[node].RightAsTarget;
104
105    /// <summary>
106    /// <para>
107    /// Sets the left using the specified node.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="node">
112    /// <para>The node.</para>
113    /// <para></para>

```

```

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

```



```

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

```

```

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

```

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

```

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

```

```

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

```

```

118 /// <para>The memory.</para>
119 /// <para></para>
120 /// </param>
121 [MethodImpl(MethodImplOptions.AggressiveInlining)]
122 protected override void SetPointers(IResizableDirectMemory memory)
123 {
124     _header = (LinksHeader<uint>*)memory.Pointer;
125     _links = (RawLink<uint>*)memory.Pointer;
126     SourcesTreeMethods = _createSourceTreeMethods();
127     TargetsTreeMethods = _createTargetTreeMethods();
128     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_links, _header);
129 }
130
131 /// <summary>
132 /// <para>
133 /// Resets the pointers.
134 /// </para>
135 /// <para></para>
136 /// </summary>
137 [MethodImpl(MethodImplOptions.AggressiveInlining)]
138 protected override void ResetPointers()
139 {
140     base.ResetPointers();
141     _links = null;
142     _header = null;
143 }
144
145 /// <summary>
146 /// <para>
147 /// Gets the header reference.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <returns>
152 /// <para>A ref links header of uint</para>
153 /// <para></para>
154 /// </returns>
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
157
158 /// <summary>
159 /// <para>
160 /// Gets the link reference using the specified link index.
161 /// </para>
162 /// <para></para>
163 /// </summary>
164 /// <param name="linkIndex">
165 /// <para>The link index.</para>
166 /// <para></para>
167 /// </param>
168 /// <returns>
169 /// <para>A ref raw link of uint</para>
170 /// <para></para>
171 /// </returns>
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 protected override ref RawLink<uint> GetLinkReference(uint linkIndex) => ref
174     ↪ _links[linkIndex];
175
176 /// <summary>
177 /// <para>
178 /// Determines whether this instance are equal.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 /// <param name="first">
183 /// <para>The first.</para>
184 /// <para></para>
185 /// </param>
186 /// <param name="second">
187 /// <para>The second.</para>
188 /// <para></para>
189 /// </param>
190 /// <returns>
191 /// <para>The bool</para>
192 /// <para></para>
193 /// </returns>
194 [MethodImpl(MethodImplOptions.AggressiveInlining)]
195 protected override bool AreEqual(uint first, uint second) => first == second;

```

```

195     /// <summary>
196     /// <para>
197     /// Determines whether this instance less than.
198     /// </para>
199     /// </summary>
200     /// <param name="first">
201     /// <para>The first.</para>
202     /// </param>
203     /// <param name="second">
204     /// <para>The second.</para>
205     /// </param>
206     /// <returns>
207     /// <para>The bool</para>
208     /// </returns>
209     [MethodImpl(MethodImplOptions.AggressiveInlining)]
210     protected override bool LessThan(uint first, uint second) => first < second;
211
212     /// <summary>
213     /// <para>
214     /// Determines whether this instance less or equal than.
215     /// </para>
216     /// </summary>
217     /// <param name="first">
218     /// <para>The first.</para>
219     /// </param>
220     /// <param name="second">
221     /// <para>The second.</para>
222     /// </param>
223     /// <returns>
224     /// <para>The bool</para>
225     /// </returns>
226     [MethodImpl(MethodImplOptions.AggressiveInlining)]
227     protected override bool LessOrEqualThan(uint first, uint second) => first <= second;
228
229     /// <summary>
230     /// <para>
231     /// Determines whether this instance greater than.
232     /// </para>
233     /// </summary>
234     /// <param name="first">
235     /// <para>The first.</para>
236     /// </param>
237     /// <param name="second">
238     /// <para>The second.</para>
239     /// </param>
240     /// <returns>
241     /// <para>The bool</para>
242     /// </returns>
243     [MethodImpl(MethodImplOptions.AggressiveInlining)]
244     protected override bool GreaterThan(uint first, uint second) => first > second;
245
246     /// <summary>
247     /// <para>
248     /// Determines whether this instance greater or equal than.
249     /// </para>
250     /// </summary>
251     /// <param name="first">
252     /// <para>The first.</para>
253     /// </param>
254     /// <param name="second">
255     /// <para>The second.</para>
256     /// </param>
257     /// <returns>
258     /// <para>The bool</para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
262
263     /// <summary>
264     /// <para>
265     /// Determines whether this instance less than.
266     /// </para>
267     /// </summary>
268     /// <param name="first">
269     /// <para>The first.</para>
270     /// </param>
271     /// <param name="second">
272     /// <para>The second.</para>
273     /// </param>
274     /// <returns>
275     /// <para>The bool</para>
276     /// </returns>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override bool LessThan(uint first, uint second) => first < second;
279
280     /// <summary>
281     /// <para>
282     /// Determines whether this instance less or equal than.
283     /// </para>
284     /// </summary>
285     /// <param name="first">
286     /// <para>The first.</para>
287     /// </param>
288     /// <param name="second">
289     /// <para>The second.</para>
290     /// </param>
291     /// <returns>
292     /// <para>The bool</para>
293     /// </returns>
294     [MethodImpl(MethodImplOptions.AggressiveInlining)]
295     protected override bool LessOrEqualThan(uint first, uint second) => first <= second;
296
297     /// <summary>
298     /// <para>
299     /// Determines whether this instance greater than.
300     /// </para>
301     /// </summary>
302     /// <param name="first">
303     /// <para>The first.</para>
304     /// </param>
305     /// <param name="second">
306     /// <para>The second.</para>
307     /// </param>
308     /// <returns>
309     /// <para>The bool</para>
310     /// </returns>
311     [MethodImpl(MethodImplOptions.AggressiveInlining)]
312     protected override bool GreaterThan(uint first, uint second) => first > second;
313
314     /// <summary>
315     /// <para>
316     /// Determines whether this instance greater or equal than.
317     /// </para>
318     /// </summary>
319     /// <param name="first">
320     /// <para>The first.</para>
321     /// </param>
322     /// <param name="second">
323     /// <para>The second.</para>
324     /// </param>
325     /// <returns>
326     /// <para>The bool</para>
327     /// </returns>
328     [MethodImpl(MethodImplOptions.AggressiveInlining)]
329     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;

```

```

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

```

```

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

```

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

```

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

```

```

11 /// </para>
12 /// <para></para>
13 /// </summary>
14 /// <seealso cref="UnusedLinksListMethods{uint}"/>
15 public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<uint>
16 {
17     private readonly RawLink<uint>* _links;
18     private readonly LinksHeader<uint>* _header;
19
20     /// <summary>
21     /// <para>
22     /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public UInt32UnusedLinksListMethods(RawLink<uint>* links, LinksHeader<uint>* header)
36         : base((byte*)links, (byte*)header)
37     {
38         _links = links;
39         _header = header;
40     }
41
42     /// <summary>
43     /// <para>
44     /// Gets the link reference using the specified link.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="link">
49     /// <para>The link.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>A ref raw link of uint</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref RawLink<uint> GetLinkReference(uint link) => ref _links[link];
58
59     /// <summary>
60     /// <para>
61     /// Gets the header reference.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <returns>
66     /// <para>A ref links header of uint</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
71 }
72 }

```

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

```

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

```



```

16 public unsafe abstract class UInt64LinksAvlBalancedTreeMethodsBase :
    ↳ LinksAvlBalancedTreeMethodsBase<ulong>
17 {
18     /// <summary>
19     /// <para>
20     /// The links.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     protected new readonly RawLink<ulong>* Links;
25     /// <summary>
26     /// <para>
27     /// The header.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     protected new readonly LinksHeader<ulong>* Header;
32
33     /// <summary>
34     /// <para>
35     /// Initializes a new <see cref="UInt64LinksAvlBalancedTreeMethodsBase"/> instance.
36     /// </para>
37     /// <para></para>
38     /// </summary>
39     /// <param name="constants">
40     /// <para>A constants.</para>
41     /// <para></para>
42     /// </param>
43     /// <param name="links">
44     /// <para>A links.</para>
45     /// <para></para>
46     /// </param>
47     /// <param name="header">
48     /// <para>A header.</para>
49     /// <para></para>
50     /// </param>
51     protected UInt64LinksAvlBalancedTreeMethodsBase(LinksConstants<ulong> constants,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header)
52     : base(constants, (byte*)links, (byte*)header)
53     {
54         Links = links;
55         Header = header;
56     }
57
58     /// <summary>
59     /// <para>
60     /// Gets the zero.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <returns>
65     /// <para>The ulong</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ulong GetZero() => 0UL;
70
71     /// <summary>
72     /// <para>
73     /// Determines whether this instance equal to zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="value">
78     /// <para>The value.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The bool</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override bool EqualToZero(ulong value) => value == 0UL;
87
88     /// <summary>
89     /// <para>
90     /// Determines whether this instance are equal.
91     /// </para>

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

548     /// <returns>
549     /// <para>A ref raw link of ulong</para>
550     /// <para></para>
551     /// </returns>
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]
553     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
554 }
555 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10         /// Represents the int 64 links recursionless size balanced tree methods base.
11         /// </para>
12         /// <para></para>
13         /// </summary>
14         /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{ulong}" />
15         public unsafe abstract class UInt64LinksRecursionlessSizeBalancedTreeMethodsBase :
16             ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<ulong>
17         {
18             /// <summary>
19             /// <para>
20             /// The links.
21             /// </para>
22             /// <para></para>
23             /// </summary>
24             protected new readonly RawLink<ulong>* Links;
25             /// <summary>
26             /// <para>
27             /// The header.
28             /// </para>
29             /// <para></para>
30             /// </summary>
31             protected new readonly LinksHeader<ulong>* Header;
32
33             /// <summary>
34             /// <para>
35             /// Initializes a new <see cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase" />
36             ↳ instance.
37             /// </para>
38             /// <para></para>
39             /// </summary>
40             /// <param name="constants">
41             /// <para>A constants.</para>
42             /// <para></para>
43             /// </param>
44             /// <param name="links">
45             /// <para>A links.</para>
46             /// <para></para>
47             /// </param>
48             /// <param name="header">
49             /// <para>A header.</para>
50             /// <para></para>
51             /// </param>
52             protected UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
53             ↳ constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
54             : base(constants, (byte*)links, (byte*)header)
55         {
56             Links = links;
57             Header = header;
58         }
59
60         /// <summary>
61         /// <para>
62         /// Gets the zero.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <returns>
67         /// <para>The ulong</para>
68         /// <para></para>
69         /// </returns>

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

### 1.103 ./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
26         /// <summary>
27         /// <para>
28         /// The header.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly LinksHeader<ulong>* Header;

```

```

31
32     /// <summary>
33     /// <para>
34     /// Initializes a new <see cref="UInt64LinksSizeBalancedTreeMethodsBase"/> instance.
35     /// </para>
36     /// </summary>
37     /// <param name="constants">
38     /// <para>A constants.</para>
39     /// </param>
40     /// <param name="links">
41     /// <para>A links.</para>
42     /// </param>
43     /// <param name="header">
44     /// <para>A header.</para>
45     /// </param>
46     /// </summary>
47     protected UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
48     ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
49     : base(constants, (byte*)links, (byte*)header)
50     {
51         Links = links;
52         Header = header;
53     }
54
55     /// <summary>
56     /// <para>
57     /// Gets the zero.
58     /// </para>
59     /// </summary>
60     /// <returns>
61     /// <para>The ulong</para>
62     /// </returns>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected override ulong GetZero() => OUL;
65
66     /// <summary>
67     /// <para>
68     /// Determines whether this instance equal to zero.
69     /// </para>
70     /// </summary>
71     /// <param name="value">
72     /// <para>The value.</para>
73     /// </param>
74     /// <returns>
75     /// <para>The bool</para>
76     /// </returns>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     protected override bool EqualToZero(ulong value) => value == OUL;
79
80     /// <summary>
81     /// <para>
82     /// Determines whether this instance are equal.
83     /// </para>
84     /// </summary>
85     /// <param name="first">
86     /// <para>The first.</para>
87     /// </param>
88     /// <param name="second">
89     /// <para>The second.</para>
90     /// </param>
91     /// <returns>
92     /// <para>The bool</para>
93     /// </returns>
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     protected override bool AreEqual(ulong first, ulong second) => first == second;
96
97
98
99
100
101
102
103
104
105
106
107

```

```

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

```

```

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

```



```

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

```

```

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

```

```

415     [MethodImpl(MethodImplOptions.AggressiveInlining)]
416     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
417 }
418 }

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

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

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

```

```

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

```

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

```

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

```



```

48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
    ↪ Links[node].LeftAsSource;

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

69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsSource =
    ↪ left;

120
121    /// <summary>
122    /// <para>

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 64 links targets avl balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
14    public unsafe class UInt64LinksTargetsAvlBalancedTreeMethods :
15        ↳ UInt64LinksAvlBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt64LinksTargetsAvlBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="header">
32        /// <para>A header.</para>
33        /// <para></para>
34        /// </param>
35        public UInt64LinksTargetsAvlBalancedTreeMethods(LinksConstants<ulong> constants,
36            ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37        { }
38
39        /// <summary>
40        /// <para>
41        /// Gets the left reference using the specified node.
42        /// </para>
43        /// <para></para>
44        /// </summary>
45        /// <param name="node">
46        /// <para>The node.</para>
47        /// <para></para>
48        /// </param>
49        /// <returns>
50        /// <para>The ref ulong</para>
51        /// <para></para>
52        /// </returns>
53        [MethodImpl(MethodImplOptions.AggressiveInlining)]
54        protected override ref ulong GetLeftReference(ulong node) => ref
55            ↳ Links[node].LeftAsTarget;
56
57        /// <summary>
58        /// <para>
59        /// Gets the right reference using the specified node.
60        /// </para>
61        /// <para></para>
62        /// </summary>
63        /// <param name="node">
64        /// <para>The node.</para>
65        /// <para></para>
66        /// </param>
67        /// <returns>
68        /// <para>The ref ulong</para>
69        /// <para></para>
70        /// </returns>
71        [MethodImpl(MethodImplOptions.AggressiveInlining)]
72        protected override ref ulong GetRightReference(ulong node) => ref
73            ↳ Links[node].RightAsTarget;
74
75        /// <summary>
76        /// <para>
77        /// Gets the left using the specified node.
78        /// </para>
79        /// <para></para>
80        /// </summary>
81        /// <param name="node">
82        /// <para>The node.</para>
83        /// <para></para>
84        /// </param>
85        /// <returns>
86        /// <para>The ref ulong</para>
87        /// <para></para>
88        /// </returns>
89        [MethodImpl(MethodImplOptions.AggressiveInlining)]
90        protected override ref ulong GetLeftReferenceUsingNode(ulong node) => ref
91            ↳ Links[node].LeftUsingNode;
92
93        /// <summary>
94        /// <para>
95        /// Gets the right using the specified node.
96        /// </para>
97        /// <para></para>
98        /// </summary>
99        /// <param name="node">
100       /// <para>The node.</para>
101       /// <para></para>
102       /// </param>
103       /// <returns>
104       /// <para>The ref ulong</para>
105       /// <para></para>
106       /// </returns>
107       [MethodImpl(MethodImplOptions.AggressiveInlining)]
108       protected override ref ulong GetRightReferenceUsingNode(ulong node) => ref
109           ↳ Links[node].RightUsingNode;
110    }
111 }

```



```

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

```

```

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

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

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

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

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

```

```

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

```

```

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

```

```

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

```

## 1.108 ./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>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The ref ulong</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref ulong GetLeftReference(ulong node) => ref
56             ↳ Links[node].LeftAsTarget;
57
58         /// <summary>
59         /// <para>
60         /// Gets the right reference using the specified node.
61         /// </para>
62         /// <para></para>
63         /// </summary>
64         /// <param name="node">
65         /// <para>The node.</para>
66         /// <para></para>
67         /// </param>
68         /// <returns>
69         /// <para>The ref ulong</para>
70         /// <para></para>
71         /// </returns>

```

```

65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
        ↳ Links[node].RightAsTarget;

69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetLeft(ulong node) => Links[node].LeftAsTarget;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsTarget;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsTarget =
        ↳ left;

120
121    /// <summary>
122    /// <para>
123    /// Sets the right using the specified node.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="node">
128    /// <para>The node.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="right">
132    /// <para>The right.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]
136    protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
        ↳ right;

137
138    /// <summary>
139    /// <para>

```

```

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

```

```

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

```

1.109 ./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 :
    ↳ UInt64LinksSizeBalancedTreeMethodsBase
15 {
16     /// <summary>
17     /// <para>
18     /// Initializes a new <see cref="UInt64LinksTargetsSizeBalancedTreeMethods"/> instance.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     public UInt64LinksTargetsSizeBalancedTreeMethods(LinksConstants<ulong> constants,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
    ↳ { }
35
36     /// <summary>
37     /// <para>
38     /// Gets the left reference using the specified node.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref ulong</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
    ↳ Links[node].LeftAsTarget;
52
53     /// <summary>
54     /// <para>
55     /// Gets the right reference using the specified node.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
    ↳ Links[node].RightAsTarget;
69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>

```

```

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

```

```

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

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

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

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

```

```

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

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

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

236    /// <returns>
237    /// <para>The bool</para>
238    /// <para></para>
239    /// </returns>
240    [MethodImpl(MethodImplOptions.AggressiveInlining)]
241    protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
242
243    /// <summary>
244    /// <para>
245    /// Determines whether this instance greater than.
246    /// </para>
247    /// <para></para>
248    /// </summary>
249    /// <param name="first">
250    /// <para>The first.</para>
251    /// <para></para>
252    /// </param>
253    /// <param name="second">
254    /// <para>The second.</para>
255    /// <para></para>
256    /// </param>
257    /// <returns>
258    /// <para>The bool</para>
259    /// <para></para>
260    /// </returns>
261    [MethodImpl(MethodImplOptions.AggressiveInlining)]
262    protected override bool GreaterThan(ulong first, ulong second) => first > second;
263
264    /// <summary>
265    /// <para>
266    /// Determines whether this instance greater or equal than.
267    /// </para>
268    /// <para></para>
269    /// </summary>
270    /// <param name="first">
271    /// <para>The first.</para>
272    /// <para></para>
273    /// </param>
274    /// <param name="second">
275    /// <para>The second.</para>
276    /// <para></para>
277    /// </param>
278    /// <returns>
279    /// <para>The bool</para>
280    /// <para></para>
281    /// </returns>
282    [MethodImpl(MethodImplOptions.AggressiveInlining)]
283    protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
284
285    /// <summary>
286    /// <para>
287    /// Gets the zero.
288    /// </para>
289    /// <para></para>
290    /// </summary>
291    /// <returns>
292    /// <para>The ulong</para>
293    /// <para></para>
294    /// </returns>
295    [MethodImpl(MethodImplOptions.AggressiveInlining)]
296    protected override ulong GetZero() => 0UL;
297
298    /// <summary>
299    /// <para>
300    /// Gets the one.
301    /// </para>
302    /// <para></para>
303    /// </summary>
304    /// <returns>
305    /// <para>The ulong</para>
306    /// <para></para>
307    /// </returns>
308    [MethodImpl(MethodImplOptions.AggressiveInlining)]
309    protected override ulong GetOne() => 1UL;
310
311    /// <summary>
312    /// <para>
313    /// Converts the to int 64 using the specified value.

```



```

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

```

```

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

```

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

```

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

```

```

48     /// <param name="link">
49     /// <para>The link.</para>
50     /// <para></para>
51     /// </param>
52     /// <returns>
53     /// <para>A ref raw link of ulong</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref _links[link];
58
59     /// <summary>
60     /// <para>
61     /// Gets the header reference.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     /// <returns>
66     /// <para>A ref links header of ulong</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
71 }
72 }

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3  using Platform.Interfaces;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.PropertyOperators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the properties operator.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}" />
16     /// <seealso cref="IProperties{TLink, TLink, TLink}" />
17     public class PropertiesOperator<TLink> : LinksOperatorBase<TLink>, IProperties<TLink, TLink,
18     ↪ TLink>
19     {
20         private static readonly EqualityComparer<TLink> _equalityComparer =
21         ↪ EqualityComparer<TLink>.Default;
22
23         /// <summary>
24         /// <para>
25         /// Initializes a new <see cref="PropertiesOperator" /> instance.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         /// <param name="links">
30         /// <para>A links.</para>
31         /// <para></para>
32         /// </param>
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public PropertiesOperator(ILinks<TLink> links) : base(links) { }
35
36         /// <summary>
37         /// <para>
38         /// Gets the value using the specified object.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="@object">
43         /// <para>The object.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="property">
47         /// <para>The property.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The link</para>
52         /// <para></para>

```

```

51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public TLink GetValue(TLink @object, TLink property)
54     {
55         var links = _links;
56         var objectProperty = links.SearchOrDefault(@object, property);
57         if (_equalityComparer.Equals(objectProperty, default))
58         {
59             return default;
60         }
61         var constants = links.Constants;
62         var any = constants.Any;
63         var query = new Link<TLink>(any, objectProperty, any);
64         var valueLink = links.SingleOrDefault(query);
65         if (valueLink == null)
66         {
67             return default;
68         }
69         return links.GetTarget(valueLink[constants.IndexPart]);
70     }
71
72     /// <summary>
73     /// <para>
74     /// Sets the value using the specified object.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="@object">
79     /// <para>The object.</para>
80     /// <para></para>
81     /// </param>
82     /// <param name="property">
83     /// <para>The property.</para>
84     /// <para></para>
85     /// </param>
86     /// <param name="value">
87     /// <para>The value.</para>
88     /// <para></para>
89     /// </param>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public void SetValue(TLink @object, TLink property, TLink value)
92     {
93         var links = _links;
94         var objectProperty = links.GetOrCreate(@object, property);
95         links.DeleteMany(links.AllIndices(links.Constants.Any, objectProperty));
96         links.GetOrCreate(objectProperty, value);
97     }
98 }
99 }

```

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

```

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

```

```

28     /// </summary>
29     /// <param name="links">
30     /// <para>A links.</para>
31     /// <para></para>
32     /// </param>
33     /// <param name="propertyMarker">
34     /// <para>A property marker.</para>
35     /// <para></para>
36     /// </param>
37     /// <param name="propertyValueMarker">
38     /// <para>A property value marker.</para>
39     /// <para></para>
40     /// </param>
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     public PropertyOperator(ILinks<TLink> links, TLink propertyMarker, TLink
43     ↪ propertyValueMarker) : base(links)
44     {
45         _propertyMarker = propertyMarker;
46         _propertyValueMarker = propertyValueMarker;
47     }
48     /// <summary>
49     /// <para>
50     /// Gets the link.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     /// <param name="link">
55     /// <para>The link.</para>
56     /// <para></para>
57     /// </param>
58     /// <returns>
59     /// <para>The link</para>
60     /// <para></para>
61     /// </returns>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public TLink Get(TLink link)
64     {
65         var property = _links.SearchOrDefault(link, _propertyMarker);
66         return GetValue(GetContainer(property));
67     }
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     private TLink GetContainer(TLink property)
70     {
71         var valueContainer = default(TLink);
72         if (_equalityComparer.Equals(property, default))
73         {
74             return valueContainer;
75         }
76         var links = _links;
77         var constants = links.Constants;
78         var countinueConstant = constants.Continue;
79         var breakConstant = constants.Break;
80         var anyConstant = constants.Any;
81         var query = new Link<TLink>(anyConstant, property, anyConstant);
82         links.Each(candidate =>
83         {
84             var candidateTarget = links.GetTarget(candidate);
85             var valueTarget = links.GetTarget(candidateTarget);
86             if (_equalityComparer.Equals(valueTarget, _propertyValueMarker))
87             {
88                 valueContainer = links.GetIndex(candidate);
89                 return breakConstant;
90             }
91             return countinueConstant;
92         }, query);
93         return valueContainer;
94     }
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     private TLink GetValue(TLink container) => _equalityComparer.Equals(container, default)
97     ↪ ? default : _links.GetTarget(container);
98     /// <summary>
99     /// <para>
100    /// Sets the link.
101    /// </para>
102    /// <para></para>
103    /// </summary>

```

```

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

```

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

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4  using Platform.Collections.Stacks;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Stacks
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the stack.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="LinksOperatorBase{TLink}" />
17     /// <seealso cref="IStack{TLink}" />
18     public class Stack<TLink> : LinksOperatorBase<TLink>, IStack<TLink>
19     {
20         private static readonly EqualityComparer<TLink> _equalityComparer =
21             ↪ EqualityComparer<TLink>.Default;
22         private readonly TLink _stack;
23
24         /// <summary>
25         /// <para>
26         /// Gets the is empty value.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         public bool IsEmpty
31         {
32             [MethodImpl(MethodImplOptions.AggressiveInlining)]
33             get => _equalityComparer.Equals(Peek(), _stack);
34         }
35
36         /// <summary>
37         /// <para>
38         /// Initializes a new <see cref="Stack" /> instance.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         /// <param name="links">
43         /// <para>A links.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="stack">
47         /// <para>A stack.</para>
48         /// <para></para>
49         /// </param>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public Stack(ILinks<TLink> links, TLink stack) : base(links) => _stack = stack;
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

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

```

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

```

```

21     /// <typeparam name="TLink">
22     /// <para>The link.</para>
23     /// <para></para>
24     /// </typeparam>
25     /// <param name="links">
26     /// <para>The links.</para>
27     /// <para></para>
28     /// </param>
29     /// <param name="stackMarker">
30     /// <para>The stack marker.</para>
31     /// <para></para>
32     /// </param>
33     /// <returns>
34     /// <para>The stack.</para>
35     /// <para></para>
36     /// </returns>
37     [MethodImpl(MethodImplOptions.AggressiveInlining)]
38     public static TLink CreateStack<TLink>(this ILinks<TLink> links, TLink stackMarker)
39     {
40         var stackPoint = links.CreatePoint();
41         var stack = links.Update(stackPoint, stackMarker, stackPoint);
42         return stack;
43     }
44 }
45 }

```

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

```

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

```



```

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

```

```

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

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

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

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

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

```

```

197         // return SyncRoot.ExecuteReadOperation(restriction, matchedHandler, substitution,
198         ↪ substitutedHandler, Unsync.Trigger);
199     //}
200 }

```

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

```

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

```

```

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     /// </summary>
110     /// <param name="links">
111     /// <para>The links.</para>
112     /// <para></para>
113     /// </param>
114     /// <param name="linkIndex">
115     /// <para>The link index.</para>
116     /// <para></para>
117     /// </param>
118     /// <param name="isElement">
119     /// <para>The is element.</para>
120     /// <para></para>
121     /// </param>
122     /// <param name="appendElement">
123     /// <para>The append element.</para>
124     /// <para></para>
125     /// </param>
126     /// <param name="renderIndex">
127     /// <para>The render index.</para>
128     /// <para></para>
129     /// </param>
130     /// <param name="renderDebug">
131     /// <para>The render debug.</para>
132     /// <para></para>
133     /// </param>
134     /// <returns>
135     /// <para>The string</para>
136     /// <para></para>
137     /// </returns>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
140     ↪ Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
141     ↪ bool renderIndex = false, bool renderDebug = false)
142     {
143         var sb = new StringBuilder();
144         var visited = new HashSet<ulong>();
145         links.AppendStructure(sb, visited, linkIndex, isElement, appendElement, renderIndex,
146         ↪ renderDebug);
147         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,
189     ↪ HashSet<ulong> visited, ulong linkIndex, Func<Link<ulong>, bool> isElement,
190     ↪ Action<StringBuilder, Link<ulong>> appendElement, bool renderIndex = false, bool
191     ↪ renderDebug = false)
192     {
193         if (sb == null)
194         {
195             throw new ArgumentNullException(nameof(sb));
196         }
197         if (linkIndex == Constants.Null || linkIndex == Constants.Any || linkIndex ==
198         ↪ Constants.Itself)
199         {
200             return;
201         }
202         if (links.Exists(linkIndex))
203         {
204             if (visited.Add(linkIndex))
205             {
206                 sb.Append('(');
207                 var link = new Link<ulong>(links.GetLink(linkIndex));
208                 if (renderIndex)
209                 {
210                     sb.Append(link.Index);
211                     sb.Append(':');
212                 }
213                 if (link.Source == link.Index)
214                 {
215                     sb.Append(link.Index);
216                 }
217                 else
218                 {
219                     var source = new Link<ulong>(links.GetLink(link.Source));
220                     if (isElement(source))

```

```

217         {
218             appendElement(sb, source);
219         }
220         else
221         {
222             links.AppendStructure(sb, visited, source.Index, isElement,
223                 ↪ appendElement, renderIndex);
224         }
225     }
226     sb.Append(' ');
227     if (link.Target == link.Index)
228     {
229         sb.Append(link.Index);
230     }
231     else
232     {
233         var target = new Link<ulong>(links.GetLink(link.Target));
234         if (isElement(target))
235         {
236             appendElement(sb, target);
237         }
238         else
239         {
240             links.AppendStructure(sb, visited, target.Index, isElement,
241                 ↪ appendElement, renderIndex);
242         }
243     }
244     sb.Append(')');
245 }
246 else
247 {
248     if (renderDebug)
249     {
250         sb.Append('*');
251     }
252     sb.Append(linkIndex);
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }

```

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

```

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

```

```

29     /// <remarks>
30     /// Альтернативные варианты хранения трансформации (элемента транзакции):
31     ///
32     /// private enum TransitionType
33     /// {
34     ///     Creation,
35     ///     UpdateOf,
36     ///     UpdateTo,
37     ///     Deletion
38     /// }
39     ///
40     /// private struct Transition
41     /// {
42     ///     public ulong TransactionId;
43     ///     public UniqueTimestamp Timestamp;
44     ///     public TransactionItemType Type;
45     ///     public Link Source;
46     ///     public Link Linker;
47     ///     public Link Target;
48     /// }
49     ///
50     /// Или
51     ///
52     /// public struct TransitionHeader
53     /// {
54     ///     public ulong TransactionIdCombined;
55     ///     public ulong TimestampCombined;
56     ///
57     ///     public ulong TransactionId
58     ///     {
59     ///         get
60     ///         {
61     ///             return (ulong) mask & TransactionIdCombined;
62     ///         }
63     ///     }
64     ///
65     ///     public UniqueTimestamp Timestamp
66     ///     {
67     ///         get
68     ///         {
69     ///             return (UniqueTimestamp)mask & TransactionIdCombined;
70     ///         }
71     ///     }
72     ///
73     ///     public TransactionItemType Type
74     ///     {
75     ///         get
76     ///         {
77     ///             // Использовать по одному биту из TransactionId и Timestamp,
78     ///             // для значения в 2 бита, которое представляет тип операции
79     ///             throw new NotImplementedException();
80     ///         }
81     ///     }
82     /// }
83     ///
84     /// private struct Transition
85     /// {
86     ///     public TransitionHeader Header;
87     ///     public Link Source;
88     ///     public Link Linker;
89     ///     public Link Target;
90     /// }
91     ///
92     /// </remarks>
93     public struct Transition : IEquatable<Transition>
94     {
95         /// <summary>
96         /// <para>
97         /// The size.
98         /// </para>
99         /// <para></para>
100        /// </summary>
101        public static readonly long Size = Structure<Transition>.Size;
102
103        /// <summary>
104        /// <para>
105        /// The transaction id.
106        /// </para>

```

```

107     /// <para></para>
108     /// </summary>
109     public readonly ulong TransactionId;
110     /// <summary>
111     /// <para>
112     /// The before.
113     /// </para>
114     /// <para></para>
115     /// </summary>
116     public readonly Link<ulong> Before;
117     /// <summary>
118     /// <para>
119     /// The after.
120     /// </para>
121     /// <para></para>
122     /// </summary>
123     public readonly Link<ulong> After;
124     /// <summary>
125     /// <para>
126     /// The timestamp.
127     /// </para>
128     /// <para></para>
129     /// </summary>
130     public readonly Timestamp Timestamp;
131
132     /// <summary>
133     /// <para>
134     /// Initializes a new <see cref="Transition"/> instance.
135     /// </para>
136     /// <para></para>
137     /// </summary>
138     /// <param name="uniqueTimestampFactory">
139     /// <para>A unique timestamp factory.</para>
140     /// <para></para>
141     /// </param>
142     /// <param name="transactionId">
143     /// <para>A transaction id.</para>
144     /// <para></para>
145     /// </param>
146     /// <param name="before">
147     /// <para>A before.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="after">
151     /// <para>A after.</para>
152     /// <para></para>
153     /// </param>
154     [MethodImpl(MethodImplOptions.AggressiveInlining)]
155     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId, Link<ulong> before, Link<ulong> after)
156     {
157         TransactionId = transactionId;
158         Before = before;
159         After = after;
160         Timestamp = uniqueTimestampFactory.Create();
161     }
162
163     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId, IList<ulong> before, IList<ulong> after) :
164         ↳ this(uniqueTimestampFactory, transactionId, new Link<ulong>(before), new
165         ↳ Link<ulong>(after)) { }
166
167     /// <summary>
168     /// <para>
169     /// Initializes a new <see cref="Transition"/> instance.
170     /// </para>
171     /// <para></para>
172     /// </summary>
173     /// <param name="uniqueTimestampFactory">
174     /// <para>A unique timestamp factory.</para>
175     /// <para></para>
176     /// </param>
177     /// <param name="transactionId">
178     /// <para>A transaction id.</para>
179     /// <para></para>
180     /// </param>
181     /// <param name="before">
182     /// <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     /// </summary>
191     /// <param name="uniqueTimestampFactory">
192     /// <para>A unique timestamp factory.</para>
193     /// </param>
194     /// <param name="transactionId">
195     /// <para>A transaction id.</para>
196     /// </param>
197     [MethodImpl(MethodImplOptions.AggressiveInlining)]
198     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
        ↳ }

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

215
216     /// <summary>
217     /// <para>
218     /// Determines whether this instance equals.
219     /// </para>
220     /// </summary>
221     /// <param name="obj">
222     /// <para>The obj.</para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     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     /// </summary>
238     /// <returns>
239     /// <para>The int</para>
240     /// </returns>
241     [MethodImpl(MethodImplOptions.AggressiveInlining)]
242     public override int GetHashCode() => (TransactionId, Before, After,
        ↳ Timestamp).GetHashCode();

245
246     /// <summary>
247     /// <para>
248     /// Determines whether this instance equals.
249     /// </para>
250     /// </summary>

```

```

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,
393     ↪ Transaction transaction)
394     {
395         layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
396         layer._currentTransactionTransitions = transaction._transitions;
397         layer._currentTransaction = transaction;

```

```

397     }
398
399     /// <summary>
400     /// <para>
401     /// Ensures the transaction allows write operations using the specified transaction.
402     /// </para>
403     /// <para></para>
404     /// </summary>
405     /// <param name="transaction">
406     /// <para>The transaction.</para>
407     /// <para></para>
408     /// </param>
409     /// <exception cref="InvalidOperationException">
410     /// <para>Transation is committed.</para>
411     /// <para></para>
412     /// </exception>
413     /// <exception cref="InvalidOperationException">
414     /// <para>Transation is reverted.</para>
415     /// <para></para>
416     /// </exception>
417     [MethodImpl(MethodImplOptions.AggressiveInlining)]
418     public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
419     {
420         if (transaction.IsReverted)
421         {
422             throw new InvalidOperationException("Transation is reverted.");
423         }
424         if (transaction.IsCommitted)
425         {
426             throw new InvalidOperationException("Transation is committed.");
427         }
428     }
429
430     /// <summary>
431     /// <para>
432     /// Disposes the manual.
433     /// </para>
434     /// <para></para>
435     /// </summary>
436     /// <param name="manual">
437     /// <para>The manual.</para>
438     /// <para></para>
439     /// </param>
440     /// <param name="wasDisposed">
441     /// <para>The was disposed.</para>
442     /// <para></para>
443     /// </param>
444     [MethodImpl(MethodImplOptions.AggressiveInlining)]
445     protected override void Dispose(bool manual, bool wasDisposed)
446     {
447         if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
448         {
449             if (!IsCommitted && !IsReverted)
450             {
451                 Revert();
452             }
453             _layer.ResetCurrentTransation();
454         }
455     }
456 }
457
458 /// <summary>
459 /// <para>
460 /// The from seconds.
461 /// </para>
462 /// <para></para>
463 /// </summary>
464 public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
465 private readonly string _logAddress;
466 private readonly FileStream _log;
467 private readonly Queue<Transition> _transitions;
468 private readonly UniqueTimestampFactory _uniqueTimestampFactory;
469 private Task _transitionsPusher;
470 private Transition _lastCommittedTransition;
471 private ulong _currentTransactionId;
472 private Queue<Transition> _currentTransactionTransitions;
473 private Transaction _currentTransaction;
474 private ulong _lastCommittedTransactionId;
475

```

```

476 /// <summary>
477 /// <para>
478 /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.
479 /// </para>
480 /// <para></para>
481 /// </summary>
482 /// <param name="links">
483 /// <para>A links.</para>
484 /// <para></para>
485 /// </param>
486 /// <param name="logAddress">
487 /// <para>A log address.</para>
488 /// <para></para>
489 /// </param>
490 /// <exception cref="ArgumentNullException">
491 /// <para></para>
492 /// <para></para>
493 /// </exception>
494 /// <exception cref="NotSupportedException">
495 /// <para>Database is damaged, autorecovery is not supported yet.</para>
496 /// <para></para>
497 /// </exception>
498 [MethodImpl(MethodImplOptions.AggressiveInlining)]
499 public UInt64LinksTransactionsLayer(ILinks<ulong> links, string logAddress)
500 : base(links)
501 {
502     if (string.IsNullOrEmpty(logAddress))
503     {
504         throw new ArgumentNullException(nameof(logAddress));
505     }
506     // В первой строке файла хранится последняя закоммиченную транзакцию.
507     // При запуске это используется для проверки удачного закрытия файла лога.
508     // In the first line of the file the last committed transaction is stored.
509     // On startup, this is used to check that the log file is successfully closed.
510     var lastCommittedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
511     var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
512     if (!lastCommittedTransition.Equals(lastWrittenTransition))
513     {
514         Dispose();
515         throw new NotSupportedException("Database is damaged, autorecovery is not
516             ↳ supported yet.");
517     }
518     if (lastCommittedTransition == default)
519     {
520         FileHelpers.WriteFirst(logAddress, lastCommittedTransition);
521     }
522     _lastCommittedTransition = lastCommittedTransition;
523     // TODO: Think about a better way to calculate or store this value
524     var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
525     _lastCommittedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
526         ↳ x.TransactionId) : 0;
527     _uniqueTimestampFactory = new UniqueTimestampFactory();
528     _logAddress = logAddress;
529     _log = FileHelpers.Append(logAddress);
530     _transitions = new Queue<Transition>();
531     _transitionsPusher = new Task(TransitionsPusher);
532     _transitionsPusher.Start();
533 }
534
535 /// <summary>
536 /// <para>
537 /// Gets the link value using the specified link.
538 /// </para>
539 /// <para></para>
540 /// </summary>
541 /// <param name="link">
542 /// <para>The link.</para>
543 /// <para></para>
544 /// </param>
545 /// <returns>
546 /// <para>A list of ulong</para>
547 /// <para></para>
548 /// </returns>
549 [MethodImpl(MethodImplOptions.AggressiveInlining)]
550 public IList<ulong> GetLinkValue(ulong link) => _links.GetLink(link);
551
552 /// <summary>
553 /// <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(ICollection<ulong> substitution, WriteHandler<ulong> handler)
566     {
567         return _links.Create(new Link<ulong>(), (before, after) =>
568         {
569             CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
570                 ↪ new Link<ulong>(before), new Link<ulong>(after)));
571             return handler(before, after);
572         });
573     }
574     /// <summary>
575     /// <para>
576     /// Updates the substitution.
577     /// </para>
578     /// <para></para>
579     /// </summary>
580     /// <param name="restriction">
581     /// <para>The substitution.</para>
582     /// <para></para>
583     /// </param>
584     /// <param name="substitution">
585     /// <para>The substitution.</para>
586     /// <para></para>
587     /// </param>
588     /// <returns>
589     /// <para>The link index.</para>
590     /// <para></para>
591     /// </returns>
592     [MethodImpl(MethodImplOptions.AggressiveInlining)]
593     public override ulong Update(ICollection<ulong> restriction, ICollection<ulong> substitution,
594         ↪ WriteHandler<ulong> handler)
595     {
596         return _links.Update(restriction, substitution, (before, after) =>
597         {
598             CommitTransition(new Transition(_uniqueTimestampFactory,
599                 ↪ _currentTransactionId, new Link<ulong>(before), new Link<ulong>(after)));
600             return handler(before, after);
601         });
602     }
603     /// <summary>
604     /// <para>
605     /// Deletes the substitution.
606     /// </para>
607     /// <para></para>
608     /// </summary>
609     /// <param name="restriction">
610     /// <para>The substitution.</para>
611     /// <para></para>
612     /// </param>
613     [MethodImpl(MethodImplOptions.AggressiveInlining)]
614     public override ulong Delete(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 ??
        ↪ _transitions;

```

```

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

```

```

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

```

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

```

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

```



```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

1  using System.IO;
2  using Xunit;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.United.Specific;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public static class ResizableDirectMemoryLinksTests
10     {
11         private static readonly LinksConstants<ulong> _constants =
12             ↪ Default<LinksConstants<ulong>>.Instance;
13
14         [Fact]
15         public static void BasicFileMappedMemoryTest()
16         {
17             var tempFilename = Path.GetTempFileName();
18             using (var memoryAdapter = new UInt64UnitedMemoryLinks(tempFilename))
19             {
20                 memoryAdapter.TestBasicMemoryOperations();
21             }
22             File.Delete(tempFilename);
23
24             [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))

```

```

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

```

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

```

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

```

## 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().Test
36             ↪ stMultipleRandomCreationsAndDeletions(100));
37             Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
38             ↪ MultipleRandomCreationsAndDeletions(100));
39             Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
40             ↪ tMultipleRandomCreationsAndDeletions(100));
41         }
42         private static void Using<TLink>(Action<ILinks<TLink>> action) where TLink : struct
43         {
44             using (var dataMemory = new HeapResizableDirectMemory())
45             using (var indexMemory = new HeapResizableDirectMemory())
46             using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory))
47             {
48                 action(memory);
49             }
50         }
51         private static void UsingWithExternalReferences<TLink>(Action<ILinks<TLink>> action)
52             ↪ where TLink : struct
53         {
54             var contants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
55             using (var dataMemory = new HeapResizableDirectMemory())
56             using (var indexMemory = new HeapResizableDirectMemory())
57             using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory,
58             ↪ SplitMemoryLinks<TLink>.DefaultLinksSizeStep, contants))
59             {
60                 action(memory);
61             }
62         }
63     }
64 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLink = System.UInt32;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryUInt32LinksTests
10     {
11         [Fact]
12         public static void CRUDTest()

```

```

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

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLink = System.UInt64;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryUInt64LinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using(links => links.TestCRUDOperations());
15         }
16
17         [Fact]
18         public static void RawNumbersCRUDTest()
19         {
20             UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
21         }
22
23         [Fact]
24         public static void MultipleRandomCreationsAndDeletionsTest()
25         {
26             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(500));
27         }
28         private static void Using(Action<ILinks<TLink>> action)
29         {
30             using (var dataMemory = new HeapResizableDirectMemory())
31             using (var indexMemory = new HeapResizableDirectMemory())
32             using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory))
33             {
34                 action(memory);
35             }
36         }
37         private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)

```

```

38     {
39         var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
40         using (var dataMemory = new HeapResizableDirectMemory())
41         using (var indexMemory = new HeapResizableDirectMemory())
42         using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
43             ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, constants))
44         {
45             action(memory);
46         }
47     }
48 }

```

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

```

1  using System.Collections.Generic;
2  using Xunit;
3  using Platform.Ranges;
4  using Platform.Numbers;
5  using Platform.Random;
6  using Platform.Setters;
7  using Platform.Converters;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public static class TestExtensions
12     {
13         public static void TestCRUDOperations<T>(this ILinks<T> links)
14         {
15             var constants = links.Constants;
16
17             var equalityComparer = EqualityComparer<T>.Default;
18
19             var zero = default(T);
20             var one = Arithmetic.Increment(zero);
21
22             // Create Link
23             Assert.True(equalityComparer.Equals(links.Count(), zero));
24
25             var setter = new Setter<T>(constants.Null);
26             links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
27
28             Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
29
30             var linkAddress = links.Create();
31
32             var link = new Link<T>(links.GetLink(linkAddress));
33
34             Assert.True(link.Count == 3);
35             Assert.True(equalityComparer.Equals(link.Index, linkAddress));
36             Assert.True(equalityComparer.Equals(link.Source, constants.Null));
37             Assert.True(equalityComparer.Equals(link.Target, constants.Null));
38
39             Assert.True(equalityComparer.Equals(links.Count(), one));
40
41             // Get first link
42             setter = new Setter<T>(constants.Null);
43             links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
44
45             Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
46
47             // Update link to reference itself
48             links.Update(linkAddress, linkAddress, linkAddress);
49
50             link = new Link<T>(links.GetLink(linkAddress));
51
52             Assert.True(equalityComparer.Equals(link.Source, linkAddress));
53             Assert.True(equalityComparer.Equals(link.Target, linkAddress));
54
55             // Update link to reference null (prepare for delete)
56             var updated = links.Update(linkAddress, constants.Null, constants.Null);
57
58             Assert.True(equalityComparer.Equals(updated, linkAddress));
59
60             link = new Link<T>(links.GetLink(linkAddress));
61
62             Assert.True(equalityComparer.Equals(link.Source, constants.Null));
63             Assert.True(equalityComparer.Equals(link.Target, constants.Null));
64
65             // Delete link
66             links.Delete(linkAddress);

```

```

67     Assert.True(equalityComparer.Equals(links.Count(), zero));
68
69     setter = new Setter<T>(constants.Null);
70     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
71
72     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
73 }
74
75 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
76 {
77     // Constants
78     var constants = links.Constants;
79     var equalityComparer = EqualityComparer<T>.Default;
80
81     var zero = default(T);
82     var one = Arithmetic.Increment(zero);
83     var two = Arithmetic.Increment(one);
84
85     var h106E = new Hybrid<T>(106L, isExternal: true);
86     var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
87     var h108E = new Hybrid<T>(-108L);
88
89     Assert.Equal(106L, h106E.AbsoluteValue);
90     Assert.Equal(107L, h107E.AbsoluteValue);
91     Assert.Equal(108L, h108E.AbsoluteValue);
92
93     // Create Link (External -> External)
94     var linkAddress1 = links.Create();
95
96     links.Update(linkAddress1, h106E, h108E);
97
98     var link1 = new Link<T>(links.GetLink(linkAddress1));
99
100     Assert.True(equalityComparer.Equals(link1.Source, h106E));
101     Assert.True(equalityComparer.Equals(link1.Target, h108E));
102
103     // Create Link (Internal -> External)
104     var linkAddress2 = links.Create();
105
106     links.Update(linkAddress2, linkAddress1, h108E);
107
108     var link2 = new Link<T>(links.GetLink(linkAddress2));
109
110     Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
111     Assert.True(equalityComparer.Equals(link2.Target, h108E));
112
113     // Create Link (Internal -> Internal)
114     var linkAddress3 = links.Create();
115
116     links.Update(linkAddress3, linkAddress1, linkAddress2);
117
118     var link3 = new Link<T>(links.GetLink(linkAddress3));
119
120     Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
121     Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
122
123     // Search for created link
124     var setter1 = new Setter<T>(constants.Null);
125     links.Each(h106E, h108E, setter1.SetAndReturnFalse);
126
127     Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
128
129     // Search for nonexistent link
130     var setter2 = new Setter<T>(constants.Null);
131     links.Each(h106E, h107E, setter2.SetAndReturnFalse);
132
133     Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
134
135     // Update link to reference null (prepare for delete)
136     var updated = links.Update(linkAddress3, constants.Null, constants.Null);
137
138     Assert.True(equalityComparer.Equals(updated, linkAddress3));
139
140     link3 = new Link<T>(links.GetLink(linkAddress3));
141
142     Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
143     Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
144
145     // Delete link

```

```

147     links.Delete(linkAddress3);
148
149     Assert.True(equalityComparer.Equals(links.Count(), two));
150
151     var setter3 = new Setter<T>(constants.Null);
152     links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
153
154     Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
155 }
156
157 public static void TestMultipleCreationsAndDeletions<TLink>(this ILinks<TLink> links,
158     ↪ int numberOfOperations)
159 {
160     for (int i = 0; i < numberOfOperations; i++)
161     {
162         links.Create();
163     }
164     for (int i = 0; i < numberOfOperations; i++)
165     {
166         links.Delete(links.Count());
167     }
168 }
169
170 public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
171     ↪ links, int maximumOperationsPerCycle)
172 {
173     var comparer = Comparer<TLink>.Default;
174     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
175     var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
176     for (var N = 1; N < maximumOperationsPerCycle; N++)
177     {
178         var random = new System.Random(N);
179         var created = 0UL;
180         var deleted = 0UL;
181         for (var i = 0; i < N; i++)
182         {
183             var linksCount = addressToUInt64Converter.Convert(links.Count());
184             var createPoint = random.NextBoolean();
185             if (linksCount >= 2 && createPoint)
186             {
187                 var linksAddressRange = new Range<ulong>(1, linksCount);
188                 TLink source = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
189                     ↪ ddressRange));
190                 TLink target = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
191                     ↪ ddressRange));
192                 ↪ //-V3086
193                 var resultLink = links.GetOrCreate(source, target);
194                 if (comparer.Compare(resultLink,
195                     ↪ uInt64ToAddressConverter.Convert(linksCount)) > 0)
196                 {
197                     created++;
198                 }
199             }
200             else
201             {
202                 links.Create();
203                 created++;
204             }
205         }
206         Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
207         for (var i = 0; i < N; i++)
208         {
209             TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
210             if (links.Exists(link))
211             {
212                 links.Delete(link);
213                 deleted++;
214             }
215         }
216         Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
217     }
218 }
219
220 }
221
222 }
223
224 }

```

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

```

1 using Platform.Data.Doublets.Memory;
2 using Platform.Data.Doublets.Memory.United.Generic;
3 using Platform.Data.Numbers.Raw;

```



```

4 using Platform.Memory;
5 using Platform.Numbers;
6 using Xunit;
7 using Xunit.Abstractions;
8 using TLink = System.UInt64;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     public class UInt64LinksExtensionsTests
13     {
14         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
15             ↪ Platform.IO.TemporaryFile());
16
17         public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename) where TLink :
18             ↪ struct
19         {
20             var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
21                 ↪ true);
22             return new UnitedMemoryLinks<TLink>(new
23                 ↪ FileMappedResizableDirectMemory(dataDBFilename),
24                 ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
25                 ↪ IndexTreeType.Default);
26         }
27
28         [Fact]
29         public void FormatStructureWithExternalReferenceTest()
30         {
31             ILinks<TLink> links = CreateLinks();
32             TLink zero = default;
33             var one = Arithmetic.Increment(zero);
34             var markerIndex = one;
35             var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
36             var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
37                 ↪ markerIndex));
38             AddressToRawNumberConverter<TLink> addressToNumberConverter = new();
39             var numberAddress = addressToNumberConverter.Convert(1);
40             var numberLink = links.GetOrCreate(numberMarker, numberAddress);
41             var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
42                 ↪ true);
43             Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
44         }
45     }
46 }

```

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

```

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

```

```

34         action(scope.Use<ILinks<TLink>>());
35     }
36 }
37 }
38 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLink = System.UInt64;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public unsafe static class UnitedMemoryUInt64LinksTests
12     {
13         [Fact]
14         public static void CRUDTest()
15         {
16             Using(links => links.TestCRUDOperations());
17         }
18
19         [Fact]
20         public static void RawNumbersCRUDTest()
21         {
22             Using(links => links.TestRawNumbersCRUDOperations());
23         }
24
25         [Fact]
26         public static void MultipleRandomCreationsAndDeletionsTest()
27         {
28             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
29         }
30         private static void Using(Action<ILinks<TLink>> action)
31         {
32             using (var scope = new Scope<Types<HeapResizableDirectMemory,
33                 ↳ UInt64UnitedMemoryLinks>>())
34             {
35                 action(scope.Use<ILinks<TLink>>());
36             }
37         }
38     }

```

## Index

`./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.cs`, 456  
`./csharp/Platform.Data.Doublets.Tests/ILinksBasicTests.cs`, 457  
`./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs`, 458  
`./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs`, 458  
`./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs`, 459  
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs`, 460  
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs`, 460  
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs`, 461  
`./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs`, 462  
`./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs`, 464  
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs`, 465  
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt64LinksTests.cs`, 466  
`./csharp/Platform.Data.Doublets/CriterionMatchers/TargetMatcher.cs`, 1  
`./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUniquenessAndUsagesResolver.cs`, 1  
`./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUsagesResolver.cs`, 2  
`./csharp/Platform.Data.Doublets/Decorators/LinksDecoratorBase.cs`, 3  
`./csharp/Platform.Data.Doublets/Decorators/LinksDisposableDecoratorBase.cs`, 5  
`./csharp/Platform.Data.Doublets/Decorators/LinksInnerReferenceExistenceValidator.cs`, 7  
`./csharp/Platform.Data.Doublets/Decorators/LinksItselfConstantToSelfReferenceResolver.cs`, 8  
`./csharp/Platform.Data.Doublets/Decorators/LinksNonExistentDependenciesCreator.cs`, 9  
`./csharp/Platform.Data.Doublets/Decorators/LinksNullConstantToSelfReferenceResolver.cs`, 10  
`./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessResolver.cs`, 11  
`./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessValidator.cs`, 12  
`./csharp/Platform.Data.Doublets/Decorators/LinksUsagesValidator.cs`, 13  
`./csharp/Platform.Data.Doublets/Decorators/LoggingDecorator.cs`, 14  
`./csharp/Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.cs`, 15  
`./csharp/Platform.Data.Doublets/Decorators/UInt32Links.cs`, 16  
`./csharp/Platform.Data.Doublets/Decorators/UInt64Links.cs`, 17  
`./csharp/Platform.Data.Doublets/Decorators/UniLinks.cs`, 19  
`./csharp/Platform.Data.Doublets/Doublet.cs`, 25  
`./csharp/Platform.Data.Doublets/DoubletComparer.cs`, 27  
`./csharp/Platform.Data.Doublets/FFI/UInt32UnitedMemoryLinks.cs`, 28  
`./csharp/Platform.Data.Doublets/FFI/UnitedMemoryLinks.cs`, 30  
`./csharp/Platform.Data.Doublets/ILinks.cs`, 39  
`./csharp/Platform.Data.Doublets/ILinksExtensions.cs`, 39  
`./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs`, 59  
`./csharp/Platform.Data.Doublets/Link.cs`, 59  
`./csharp/Platform.Data.Doublets/LinkExtensions.cs`, 67  
`./csharp/Platform.Data.Doublets/LinksOperatorBase.cs`, 68  
`./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs`, 68  
`./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs`, 69  
`./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs`, 70  
`./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs`, 71  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 73  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs`, 80  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 86  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 90  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 94  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 98  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 102  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs`, 107  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesLinkedListMethods.cs`, 113  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 118  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs`, 122  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 125  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs`, 129  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs`, 133  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs`, 136  
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs`, 154  
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs`, 157  
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs`, 158  
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 160  
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase.cs`, 166  
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 172  
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 176

./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 180  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs, 184  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 188  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 193  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 199  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 200  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 204  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 207  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 211  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 215  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs, 221  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 222  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase.cs, 228  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 234  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods.cs, 238  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 242  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethods.cs, 246  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 249  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.cs, 255  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 261  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 262  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 265  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 269  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 273  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 276  
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 283  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 284  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 293  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 299  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 306  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 311  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 315  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 319  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 324  
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 328  
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs, 331  
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs, 334  
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs, 347  
./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs, 350  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 352  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 357  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 363  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs, 366  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 370  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 374  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 378  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 383  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 384  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 392  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 397  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 403  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 408  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 412  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 415  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 421  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 424  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 428  
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs, 434  
./csharp/Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs, 435  
./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs, 436  
./csharp/Platform.Data.Doublets/Stacks/Stack.cs, 438  
./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs, 439

./csharp/Platform.Data.Doublets/SynchronizedLinks.cs, 440  
./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs, 443  
./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 446