

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
22        private readonly TLink _targetToMatch;
23
24        /// <summary>
25        /// <para>
26        /// Initializes a new <see cref="TargetMatcher"/> instance.
27        /// </para>
28        /// <para></para>
29        /// </summary>
30        /// <param name="links">
31        /// <para>A links.</para>
32        /// <para></para>
33        /// </param>
34        /// <param name="targetToMatch">
35        /// <para>A target to match.</para>
36        /// <para></para>
37        /// </param>
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        public TargetMatcher(ILinks<TLink> links, TLink targetToMatch) : base(links) =>
40            ↳ _targetToMatch = targetToMatch;
41
42        /// <summary>
43        /// <para>
44        /// Determines whether this instance is matched.
45        /// </para>
46        /// <para></para>
47        /// </summary>
48        /// <param name="link">
49        /// <para>The link.</para>
50        /// <para></para>
51        /// </param>
52        /// <returns>
53        /// <para>The bool</para>
54        /// <para></para>
55        /// </returns>
56        [MethodImpl(MethodImplOptions.AggressiveInlining)]
57        public bool IsMatched(TLink link) => _equalityComparer.Equals(_links.GetTarget(link),
58            ↳ _targetToMatch);
59    }
60 }
```

1.2 ./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUniquenessAndUsagesResolver.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.Decorators
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links cascade uniqueness and usages resolver.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="LinksUniquenessResolver{TLink}"/>
14    public class LinksCascadeUniquenessAndUsagesResolver<TLink> : LinksUniquenessResolver<TLink>
15    {
16    }
```

```

16     /// <summary>
17     /// <para>
18     /// Initializes a new <see cref="LinksCascadeUniquenessAndUsagesResolver"/> 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 LinksCascadeUniquenessAndUsagesResolver(ILinks<TLink> links) : base(links) { }
28
29     /// <summary>
30     /// <para>
31     /// Resolves the address change conflict using the specified old link address.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     /// <param name="oldLinkAddress">
36     /// <para>The old link address.</para>
37     /// <para></para>
38     /// </param>
39     /// <param name="newLinkAddress">
40     /// <para>The new link address.</para>
41     /// <para></para>
42     /// </param>
43     /// <returns>
44     /// <para>The link</para>
45     /// <para></para>
46     /// </returns>
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     protected override TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
49     ↪ newLinkAddress)
50     {
51         // Use Facade (the last decorator) to ensure recursion working correctly
52         _facade.MergeUsages(oldLinkAddress, newLinkAddress);
53         return base.ResolveAddressChangeConflict(oldLinkAddress, newLinkAddress);
54     }
55 }

```

1.3 ./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUsagesResolver.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.Decorators
7 {
8     /// <remarks>
9     /// <para>Must be used in conjunction with NonNullContentsLinkDeletionResolver.</para>
10    /// <para>Должен использоваться вместе с NonNullContentsLinkDeletionResolver.</para>
11    /// </remarks>
12    public class LinksCascadeUsagesResolver<TLink> : LinksDecoratorBase<TLink>
13    {
14        /// <summary>
15        /// <para>
16        /// Initializes a new <see cref="LinksCascadeUsagesResolver"/> instance.
17        /// </para>
18        /// <para></para>
19        /// </summary>
20        /// <param name="links">
21        /// <para>A links.</para>
22        /// <para></para>
23        /// </param>
24        [MethodImpl(MethodImplOptions.AggressiveInlining)]
25        public LinksCascadeUsagesResolver(ILinks<TLink> links) : base(links) { }
26
27        /// <summary>
28        /// <para>
29        /// Deletes the restrictions.
30        /// </para>
31        /// <para></para>
32        /// </summary>
33        /// <param name="restrictions">
34        /// <para>The restrictions.</para>
35        /// <para></para>

```

```

36     /// </param>
37     [MethodImpl(MethodImplOptions.AggressiveInlining)]
38     public override void Delete(ICollection<TLink> restrictions)
39     {
40         var linkIndex = restrictions[_constants.IndexPart];
41         // Use Facade (the last decorator) to ensure recursion working correctly
42         _facade.DeleteAllUsages(linkIndex);
43         _links.Delete(linkIndex);
44     }
45 }
46 }

```

1.4 ./csharp/Platform.Data.Doublets/Decorators/LinksDecoratorBase.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.Decorators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the links decorator base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}" />
16     /// <seealso cref="ILinks{TLink}" />
17     public abstract class LinksDecoratorBase<TLink> : LinksOperatorBase<TLink>, ILinks<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// The constants.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected readonly LinksConstants<TLink> _constants;
26
27         /// <summary>
28         /// <para>
29         /// Gets the constants value.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         public LinksConstants<TLink> Constants
34         {
35             [MethodImpl(MethodImplOptions.AggressiveInlining)]
36             get => _constants;
37         }
38
39         /// <summary>
40         /// <para>
41         /// The facade.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         protected ILinks<TLink> _facade;
46
47         /// <summary>
48         /// <para>
49         /// Gets or sets the facade value.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         public ILinks<TLink> Facade
54         {
55             [MethodImpl(MethodImplOptions.AggressiveInlining)]
56             get => _facade;
57             [MethodImpl(MethodImplOptions.AggressiveInlining)]
58             set
59             {
60                 _facade = value;
61                 if (_links is LinksDecoratorBase<TLink> decorator)
62                 {
63                     decorator.Facade = value;
64                 }
65             }
66         }
67     }
68 }

```

```

67
68     /// <summary>
69     /// <para>
70     /// Initializes a new <see cref="LinksDecoratorBase"/> instance.
71     /// </para>
72     /// <para></para>
73     /// </summary>
74     /// <param name="links">
75     /// <para>A links.</para>
76     /// <para></para>
77     /// </param>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     protected LinksDecoratorBase(ILinks<TLink> links) : base(links)
80     {
81         _constants = links.Constants;
82         Facade = this;
83     }
84
85     /// <summary>
86     /// <para>
87     /// Counts the restrictions.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="restrictions">
92     /// <para>The restrictions.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The link</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    public virtual TLink Count(IList<TLink> restrictions) => _links.Count(restrictions);
101
102    /// <summary>
103    /// <para>
104    /// Eaches the handler.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="handler">
109    /// <para>The handler.</para>
110    /// <para></para>
111    /// </param>
112    /// <param name="restrictions">
113    /// <para>The restrictions.</para>
114    /// <para></para>
115    /// </param>
116    /// <returns>
117    /// <para>The link</para>
118    /// <para></para>
119    /// </returns>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
122    ↪ => _links.Each(handler, restrictions);
123
124    /// <summary>
125    /// <para>
126    /// Creates the restrictions.
127    /// </para>
128    /// <para></para>
129    /// </summary>
130    /// <param name="restrictions">
131    /// <para>The restrictions.</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> restrictions) => _links.Create(restrictions);
140
141    /// <summary>
142    /// <para>
143    /// Updates the restrictions.
144    /// </para>

```

```

144     /// <para></para>
145     /// </summary>
146     /// <param name="restrictions">
147     /// <para>The restrictions.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="substitution">
151     /// <para>The substitution.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
160         ↪ _links.Update(restrictions, substitution);
161
162     /// <summary>
163     /// <para>
164     /// Deletes the restrictions.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="restrictions">
169     /// <para>The restrictions.</para>
170     /// <para></para>
171     /// </param>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     public virtual void Delete(IList<TLink> restrictions) => _links.Delete(restrictions);
174 }

```

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" />
28         protected class DisposableWithMultipleCallsAllowed : Disposable
29         {
30             /// <summary>
31             /// <para>
32             /// Initializes a new <see cref="DisposableWithMultipleCallsAllowed" /> instance.
33             /// </para>
34             /// <para></para>
35             /// </summary>
36             /// <param name="disposal">
37             /// <para>A disposal.</para>
38             /// <para></para>
39             /// </param>
40             [MethodImpl(MethodImplOptions.AggressiveInlining)]
41             public DisposableWithMultipleCallsAllowed(Disposal disposal) : base(disposal) { }
42
43             /// <summary>
44             /// <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     {
76         ~LinksDisposableDecoratorBase() => Disposable.Destruct();
77
78         /// <summary>
79         /// <para>
80         /// Disposes this instance.
81         /// </para>
82         /// <para></para>
83         /// </summary>
84         [MethodImpl(MethodImplOptions.AggressiveInlining)]
85         public void Dispose() => Disposable.Dispose();
86
87         /// <summary>
88         /// <para>
89         /// Disposes the manual.
90         /// </para>
91         /// <para></para>
92         /// </summary>
93         /// <param name="manual">
94         /// <para>The manual.</para>
95         /// <para></para>
96         /// </param>
97         /// <param name="wasDisposed">
98         /// <para>The was disposed.</para>
99         /// <para></para>
100        /// </param>
101        [MethodImpl(MethodImplOptions.AggressiveInlining)]
102        protected virtual void Dispose(bool manual, bool wasDisposed)
103        {
104            if (!wasDisposed)
105            {
106                _links.DisposeIfPossible();
107            }
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
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Decorators
8 {

```

```

9 // TODO: Make LinksExternalReferenceValidator. A layer that checks each link to exist or to
10 ↪ be external (hybrid link's raw number).
11 /// <summary>
12 /// <para>
13 /// Represents the links inner reference existence validator.
14 /// </para>
15 /// </summary>
16 /// <seealso cref="LinksDecoratorBase{TLink}"/>
17 public class LinksInnerReferenceExistenceValidator<TLink> : LinksDecoratorBase<TLink>
18 {
19     /// <summary>
20     /// <para>
21     /// Initializes a new <see cref="LinksInnerReferenceExistenceValidator"/> 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 LinksInnerReferenceExistenceValidator(ILinks<TLink> links) : base(links) { }
31
32     /// <summary>
33     /// <para>
34     /// Eaches the handler.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="handler">
39     /// <para>The handler.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="restrictions">
43     /// <para>The restrictions.</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 Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
52     {
53         var links = _links;
54         links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
55         return links.Each(handler, restrictions);
56     }
57
58     /// <summary>
59     /// <para>
60     /// Updates the restrictions.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="restrictions">
65     /// <para>The restrictions.</para>
66     /// <para></para>
67     /// </param>
68     /// <param name="substitution">
69     /// <para>The substitution.</para>
70     /// <para></para>
71     /// </param>
72     /// <returns>
73     /// <para>The link</para>
74     /// <para></para>
75     /// </returns>
76     [MethodImpl(MethodImplOptions.AggressiveInlining)]
77     public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
78     {
79         // TODO: Possible values: null, ExistentLink or NonExistentHybrid(ExternalReference)
80         var links = _links;
81         links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
82         links.EnsureInnerReferenceExists(substitution, nameof(substitution));
83         return links.Update(restrictions, substitution);
84     }
85 }

```

```

86     /// <summary>
87     /// <para>
88     /// Deletes the restrictions.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="restrictions">
93     /// <para>The restrictions.</para>
94     /// <para></para>
95     /// </param>
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public override void Delete(ICollection<TLink> restrictions)
98     {
99         var link = restrictions[_constants.IndexPart];
100         var links = _links;
101         links.EnsureLinkExists(link, nameof(link));
102         links.Delete(link);
103     }
104 }
105 }

```

1.7 ./csharp/Platform.Data.Doublets/Decorators/LinksItselfConstantToSelfReferenceResolver.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.Decorators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the links itself constant to self reference resolver.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksDecoratorBase<TLink>" />
16     public class LinksItselfConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
19             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="restrictions">
45         /// <para>The restrictions.</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(Func<ICollection<TLink>, TLink> handler, ICollection<TLink> restrictions)
54         {
55             var constants = _constants;
56             var itselfConstant = constants.Itself;

```



```

56         if (!_equalityComparer.Equals(constants.Any, itselfConstant) &&
57             ↪ restrictions.Contains(itselfConstant))
58         {
59             // Itself constant is not supported for Each method right now, skipping execution
60             return constants.Continue;
61         }
62         return _links.Each(handler, restrictions);
63     }
64     /// <summary>
65     /// <para>
66     /// Updates the restrictions.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <param name="restrictions">
71     /// <para>The restrictions.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="substitution">
75     /// <para>The substitution.</para>
76     /// <para></para>
77     /// </param>
78     /// <returns>
79     /// <para>The link</para>
80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
84         ↪ _links.Update(restrictions, _links.ResolveConstantAsSelfReference(_constants.Itself,
85         ↪ restrictions, substitution));
86     }
87 }

```

1.8 ./csharp/Platform.Data.Doublets/Decorators/LinksNonExistentDependenciesCreator.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.Decorators
7  {
8      /// <remarks>
9      /// Not practical if newSource and newTarget are too big.
10     /// To be able to use practical version we should allow to create link at any specific
11     ↪ location inside ResizableDirectMemoryLinks.
12     /// This in turn will require to implement not a list of empty links, but a list of ranges
13     ↪ to store it more efficiently.
14     /// </remarks>
15     public class LinksNonExistentDependenciesCreator<TLink> : LinksDecoratorBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksNonExistentDependenciesCreator"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </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 restrictions.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="restrictions">
37         /// <para>The restrictions.</para>
38         /// <para></para>
39         /// </param>
40         /// <param name="substitution">
41         /// <para>The substitution.</para>
42         /// <para></para>
43         /// </param>

```

```

42     /// <returns>
43     /// <para>The link</para>
44     /// <para></para>
45     /// </returns>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
48     {
49         var constants = _constants;
50         var links = _links;
51         links.EnsureCreated(substitution[constants.SourcePart],
52             ↪ substitution[constants.TargetPart]);
53         return links.Update(restrictions, substitution);
54     }
55 }

```

1.9 ./csharp/Platform.Data.Doublets/Decorators/LinksNullConstantToSelfReferenceResolver.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.Decorators
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the links null constant to self reference resolver.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksDecoratorBase{TLink}" />
15     public class LinksNullConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksNullConstantToSelfReferenceResolver" /> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="links">
24         /// <para>A links.</para>
25         /// <para></para>
26         /// </param>
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public LinksNullConstantToSelfReferenceResolver(ILinks<TLink> links) : base(links) { }
29
30         /// <summary>
31         /// <para>
32         /// Creates the restrictions.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="restrictions">
37         /// <para>The restrictions.</para>
38         /// <para></para>
39         /// </param>
40         /// <returns>
41         /// <para>The link</para>
42         /// <para></para>
43         /// </returns>
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         public override TLink Create(IList<TLink> restrictions) => _links.CreatePoint();
46
47         /// <summary>
48         /// <para>
49         /// Updates the restrictions.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="restrictions">
54         /// <para>The restrictions.</para>
55         /// <para></para>
56         /// </param>
57         /// <param name="substitution">
58         /// <para>The substitution.</para>
59         /// <para></para>
60         /// </param>
61         /// <returns>

```

```

62     /// <para>The link</para>
63     /// <para></para>
64     /// </returns>
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
        ↪ _links.Update(restrictions, _links.ResolveConstantAsSelfReference(_constants.Null,
        ↪ restrictions, substitution));
67 }
68 }

```

1.10 ./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessResolver.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.Decorators
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the links uniqueness resolver.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksDecoratorBase{TLink}" />
15     public class LinksUniquenessResolver<TLink> : LinksDecoratorBase<TLink>
16     {
17         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
18
19         /// <summary>
20         /// <para>
21         /// Initializes a new <see cref="LinksUniquenessResolver" /> 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 LinksUniquenessResolver(ILinks<TLink> links) : base(links) { }
31
32         /// <summary>
33         /// <para>
34         /// Updates the restrictions.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="restrictions">
39         /// <para>The restrictions.</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> restrictions, IList<TLink> substitution)
52         {
53             var constants = _constants;
54             var links = _links;
55             var newLinkAddress = links.SearchOrDefault(substitution[constants.SourcePart],
                ↪ substitution[constants.TargetPart]);
56             if (_equalityComparer.Equals(newLinkAddress, default))
57             {
58                 return links.Update(restrictions, substitution);
59             }
60             return ResolveAddressChangeConflict(restrictions[constants.IndexPart],
                ↪ newLinkAddress);
61         }
62
63         /// <summary>
64         /// <para>

```

```

65     /// Resolves the address change conflict using the specified old link address.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="oldLinkAddress">
70     /// <para>The old link address.</para>
71     /// <para></para>
72     /// </param>
73     /// <param name="newLinkAddress">
74     /// <para>The new link address.</para>
75     /// <para></para>
76     /// </param>
77     /// <returns>
78     /// <para>The new link address.</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected virtual TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
    ↪ newLinkAddress)
83     {
84         if (!_equalityComparer.Equals(oldLinkAddress, newLinkAddress) &&
    ↪ _links.Exists(oldLinkAddress))
85         {
86             _facade.Delete(oldLinkAddress);
87         }
88         return newLinkAddress;
89     }
90 }
91 }

```

1.11 ./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessValidator.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.Decorators
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the links uniqueness validator.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksDecoratorBase{TLink}" />
15     public class LinksUniquenessValidator<TLink> : LinksDecoratorBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksUniquenessValidator" /> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="links">
24         /// <para>A links.</para>
25         /// <para></para>
26         /// </param>
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public LinksUniquenessValidator(ILinks<TLink> links) : base(links) { }
29
30         /// <summary>
31         /// <para>
32         /// Updates the restrictions.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="restrictions">
37         /// <para>The restrictions.</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(ILink<TLink> restrictions, IList<TLink> substitution)
50     {
51         var links = _links;
52         var constants = _constants;
53         links.EnsureDoesNotExists(substitution[constants.SourcePart],
54             ↳ substitution[constants.TargetPart]);
55         return links.Update(restrictions, substitution);
56     }
57 }

```

1.12 ./csharp/Platform.Data.Doublets/Decorators/LinksUsagesValidator.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.Decorators
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the links usages validator.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksDecoratorBase{TLink}"/>
15     public class LinksUsagesValidator<TLink> : LinksDecoratorBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="LinksUsagesValidator"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="links">
24         /// <para>A links.</para>
25         /// <para></para>
26         /// </param>
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public LinksUsagesValidator(ILinks<TLink> links) : base(links) { }
29
30         /// <summary>
31         /// <para>
32         /// Updates the restrictions.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="restrictions">
37         /// <para>The restrictions.</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(ILink<TLink> restrictions, IList<TLink> substitution)
50         {
51             var links = _links;
52             links.EnsureNoUsages(restrictions[_constants.IndexPart]);
53             return links.Update(restrictions, substitution);
54         }
55
56         /// <summary>
57         /// <para>
58         /// Deletes the restrictions.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         /// <param name="restrictions">
63         /// <para>The restrictions.</para>
64         /// <para></para>
65         /// </param>
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

67     public override void Delete(ICollection<TLink> restrictions)
68     {
69         var link = restrictions[_constants.IndexPart];
70         var links = _links;
71         links.EnsureNoUsages(link);
72         links.Delete(link);
73     }
74 }
75 }

```

1.13 ./csharp/Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.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.Decorators
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the non null contents link deletion resolver.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksDecoratorBase{TLink}" />
15     public class NonNullContentsLinkDeletionResolver<TLink> : LinksDecoratorBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="NonNullContentsLinkDeletionResolver" /> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="links">
24         /// <para>A links.</para>
25         /// <para></para>
26         /// </param>
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public NonNullContentsLinkDeletionResolver(ICollection<TLink> links) : base(links) { }
29
30         /// <summary>
31         /// <para>
32         /// Deletes the restrictions.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="restrictions">
37         /// <para>The restrictions.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public override void Delete(ICollection<TLink> restrictions)
42         {
43             var linkIndex = restrictions[_constants.IndexPart];
44             var links = _links;
45             links.EnforceResetValues(linkIndex);
46             links.Delete(linkIndex);
47         }
48     }
49 }

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
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.Decorators
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 links.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksDisposableDecoratorBase{TLink}" />
16     public class UInt32Links : LinksDisposableDecoratorBase<TLink>
17     {

```

```

18     /// <summary>
19     /// <para>
20     /// Initializes a new <see cref="UInt32Links"/> instance.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     /// <param name="links">
25     /// <para>A links.</para>
26     /// <para></para>
27     /// </param>
28     [MethodImpl(MethodImplOptions.AggressiveInlining)]
29     public UInt32Links(ILinks<TLink> links) : base(links) { }
30
31     /// <summary>
32     /// <para>
33     /// Creates the restrictions.
34     /// </para>
35     /// <para></para>
36     /// </summary>
37     /// <param name="restrictions">
38     /// <para>The restrictions.</para>
39     /// <para></para>
40     /// </param>
41     /// <returns>
42     /// <para>The link</para>
43     /// <para></para>
44     /// </returns>
45     [MethodImpl(MethodImplOptions.AggressiveInlining)]
46     public override TLink Create(ICollection<TLink> restrictions) => _links.CreatePoint();
47
48     /// <summary>
49     /// <para>
50     /// Updates the restrictions.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     /// <param name="restrictions">
55     /// <para>The restrictions.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="substitution">
59     /// <para>The substitution.</para>
60     /// <para></para>
61     /// </param>
62     /// <returns>
63     /// <para>The link</para>
64     /// <para></para>
65     /// </returns>
66     [MethodImpl(MethodImplOptions.AggressiveInlining)]
67     public override TLink Update(ICollection<TLink> restrictions, ICollection<TLink> substitution)
68     {
69         var constants = _constants;
70         var indexPartConstant = constants.IndexPart;
71         var sourcePartConstant = constants.SourcePart;
72         var targetPartConstant = constants.TargetPart;
73         var nullConstant = constants.Null;
74         var itselfConstant = constants.Itself;
75         var existedLink = nullConstant;
76         var updatedLink = restrictions[indexPartConstant];
77         var newSource = substitution[sourcePartConstant];
78         var newTarget = substitution[targetPartConstant];
79         var links = _links;
80         if (newSource != itselfConstant && newTarget != itselfConstant)
81         {
82             existedLink = links.SearchOrDefault(newSource, newTarget);
83         }
84         if (existedLink == nullConstant)
85         {
86             var before = links.GetLink(updatedLink);
87             if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
88                 ↪ newTarget)
89             {
90                 links.Update(updatedLink, newSource == itselfConstant ? updatedLink :
91                     ↪ newSource,
92                             newTarget == itselfConstant ? updatedLink :
93                     ↪ newTarget);
94             }
95         }
96         return updatedLink;
97     }

```

```

93     }
94     else
95     {
96         return _facade.MergeAndDelete(updatedLink, existedLink);
97     }
98 }
99
100 /// <summary>
101 /// <para>
102 /// Deletes the restrictions.
103 /// </para>
104 /// <para></para>
105 /// </summary>
106 /// <param name="restrictions">
107 /// <para>The restrictions.</para>
108 /// <para></para>
109 /// </param>
110 [MethodImpl(MethodImplOptions.AggressiveInlining)]
111 public override void Delete(ICollection<TLink> restrictions)
112 {
113     var linkIndex = restrictions[_constants.IndexPart];
114     var links = _links;
115     links.EnforceResetValues(linkIndex);
116     _facade.DeleteAllUsages(linkIndex);
117     links.Delete(linkIndex);
118 }
119 }
120 }

```

1.15 ./csharp/Platform.Data.Doublets/Decorators/UInt64Links.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.Decorators
7 {
8     /// <summary>
9     /// <para>Represents a combined decorator that implements the basic logic for interacting
10     ↪ with the links storage for links with addresses represented as <see cref="System.UInt64"
11     ↪ >.</para>
12     /// <para>Представляет комбинированный декоратор, реализующий основную логику по
13     ↪ взаимодействию с хранилищем связей, для связей с адресами представленными в виде <see
14     ↪ cref="System.UInt64">.</para>
15     /// </summary>
16     /// <remarks>
17     /// Возможные оптимизации:
18     /// Объединение в одном поле Source и Target с уменьшением до 32 бит.
19     /// + меньше объём БД
20     /// - меньше производительность
21     /// - больше ограничение на количество связей в БД)
22     /// Ленивое хранение размеров поддеревьев (расчитываемое по мере использования БД)
23     /// + меньше объём БД
24     /// - больше сложность
25     ///
26     /// Текущее теоретическое ограничение на индекс связи, из-за использования 5 бит в размере
27     ↪ поддеревьев для AVL баланса и флагов нитей: 2 в степени(64 минус 5 равно 59 ) равно 576
28     ↪ 460 752 303 423 488
29     /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами
30     ↪ (битовыми строками) - вариант матрицы (выстраиваемой лениво).
31     ///
32     /// Решить отключать ли проверки при компиляции под Release. Т.е. исключения будут
33     ↪ выбрасываться только при #if DEBUG
34     /// </remarks>
35     public class UInt64Links : LinksDisposableDecoratorBase<ulong>
36     {
37         /// <summary>
38         /// <para>
39         /// Initializes a new <see cref="UInt64Links"/> instance.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         /// <param name="links">
44         /// <para>A links.</para>
45         /// <para></para>
46         /// </param>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public UInt64Links(ILinks<ulong> links) : base(links) { }

```



```

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

```

```

116     /// </summary>
117     /// <param name="restrictions">
118     /// <para>The restrictions.</para>
119     /// <para></para>
120     /// </param>
121     [MethodImpl(MethodImplOptions.AggressiveInlining)]
122     public override void Delete(ICollection<ulong> restrictions)
123     {
124         var linkIndex = restrictions[_constants.IndexPart];
125         var links = _links;
126         links.EnforceResetValues(linkIndex);
127         _facade.DeleteAllUsages(linkIndex);
128         links.Delete(linkIndex);
129     }
130 }
131 }

```

1.16 ./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
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Decorators
11 {
12     /// <remarks>
13     /// What does empty pattern (for condition or substitution) mean? Nothing or Everything?
14     /// Now we go with nothing. And nothing is something one, but empty, and cannot be changed
15     /// ↪ by itself. But can cause creation (update from nothing) or deletion (update to nothing).
16     ///
17     /// TODO: Decide to change to IDoubletLinks or not to change. (Better to create
18     /// ↪ DefaultUniLinksBase, that contains logic itself and can be implemented using both
19     /// ↪ IDoubletLinks and ILinks.)
20     /// </remarks>
21     internal class UniLinks<TLink> : LinksDecoratorBase<TLink>, IUniLinks<TLink>
22     {
23         private static readonly EqualityComparer<TLink> _equalityComparer =
24             ↪ EqualityComparer<TLink>.Default;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UniLinks"/> instance.
29         /// </para>
30         /// </summary>
31         /// <param name="links">
32         /// <para>A links.</para>
33         /// <para></para>
34         /// </param>
35         public UniLinks(ILinks<TLink> links) : base(links) { }
36
37         private struct Transition
38         {
39             /// <summary>
40             /// <para>
41             /// The before.
42             /// </para>
43             /// </summary>
44             public IList<TLink> Before;
45
46             /// <summary>
47             /// <para>
48             /// The after.
49             /// </para>
50             /// </summary>
51             public IList<TLink> After;
52
53             /// <summary>
54             /// <para>
55             /// Initializes a new <see cref="Transition"/> instance.
56             /// </para>
57             /// </summary>
58             /// <param name="before">

```

```

58     /// <para>A before.</para>
59     /// <para></para>
60     /// </param>
61     /// <param name="after">
62     /// <para>A after.</para>
63     /// <para></para>
64     /// </param>
65     public Transition(IList<TLink> before, IList<TLink> after)
66     {
67         Before = before;
68         After = after;
69     }
70
71
72     //public static readonly TLink NullConstant = Use<LinksConstants<TLink>>.Single.Null;
73     //public static readonly IReadOnlyList<TLink> NullLink = new
74     ↪     ReadOnlyCollection<TLink>(new List<TLink> { NullConstant, NullConstant, NullConstant
75     ↪     });
76
77     // TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
78     ↪     (Links-Expression)
79     /// <summary>
80     /// <para>
81     /// Triggers the restriction.
82     /// </para>
83     /// <para></para>
84     /// </summary>
85     /// <param name="restriction">
86     /// <para>The restriction.</para>
87     /// <para></para>
88     /// </param>
89     /// <param name="matchedHandler">
90     /// <para>The matched handler.</para>
91     /// <para></para>
92     /// </param>
93     /// <param name="substitution">
94     /// <para>The substitution.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="substitutedHandler">
98     /// <para>The substituted handler.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The link</para>
103    /// <para></para>
104    /// </returns>
105    public TLink Trigger(IList<TLink> restriction, Func<IList<TLink>, IList<TLink>, TLink>
106    ↪     matchedHandler, IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
107    ↪     substitutedHandler)
108    {
109        ///List<Transition> transitions = null;
110        ///if (!restriction.IsNullOrEmpty())
111        ///{
112        ///    // Есть причина делать проход (чтение)
113        ///    if (matchedHandler != null)
114        ///    {
115        ///        if (!substitution.IsNullOrEmpty())
116        ///        {
117        ///            // restriction => { 0, 0, 0 } | { 0 } // Create
118        ///            // substitution => { itself, 0, 0 } | { itself, itself, itself } //
119        ↪            Create / Update
120        ///            // substitution => { 0, 0, 0 } | { 0 } // Delete
121        ///            transitions = new List<Transition>();
122        ///            if (Equals(substitution[Constants.IndexPart], Constants.Null))
123        ///            {
124        ///                // If index is Null, that means we always ignore every other
125        ↪                value (they are also Null by definition)
126        ///                var matchDecision = matchedHandler(, NullLink);
127        ///                if (Equals(matchDecision, Constants.Break))
128        ///                    return false;
129        ///                if (!Equals(matchDecision, Constants.Skip))
130        ///                    transitions.Add(new Transition(matchedLink, newValue));
131        ///            }
132        ///        }
133        ///    }
134        ///    else
135        ///    {
136        ///        Func<T, bool> handler;
137        ///        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 else
147 {
148     Func<T, bool> handler = link =>
149     {
150         var matchedLink = Memory.GetLinkValue(link);
151         var matchDecision = matchedHandler(matchedLink, matchedLink);
152         return !Equals(matchDecision, Constants.Break);
153     };
154     if (!Memory.Each(handler, restriction))
155         return Constants.Break;
156 }
157 }
158 else
159 {
160     if (substitution != null)
161     {
162         transitions = new List<IList<T>>();
163         Func<T, bool> handler = link =>
164         {
165             var matchedLink = Memory.GetLinkValue(link);
166             transitions.Add(matchedLink);
167             return true;
168         };
169         if (!Memory.Each(handler, restriction))
170             return Constants.Break;
171     }
172     else
173     {
174         return Constants.Continue;
175     }
176 }
177 }
178 }
179 }
180 // Есть причина делать замену (запись)
181 if (substitutedHandler != null)
182 {
183 }
184 else
185 {
186 }
187 }
188 }
189 return Constants.Continue;
190 //if (restriction.IsNullOrEmpty()) // Create
191 //{
192 //    substitution[Constants.IndexPart] = Memory.AllocateLink();
193 //    Memory.SetLinkValue(substitution);
194 //}
195 //else if (substitution.IsNullOrEmpty()) // Delete
196 //{
197 //    Memory.FreeLink(restriction[Constants.IndexPart]);
198 //}
199 //else if (restriction.EqualTo(substitution)) // Read or ("repeat" the state) // Each
200 //{
201 //    // No need to collect links to list

```

```

202 // // Skip == Continue
203 // // No need to check substitutedHandler
204 // if (!Memory.Each(link => !Equals(matchedHandler(Memory.GetLinkValue(link)),
    ↳ Constants.Break), restriction))
205 //     return Constants.Break;
206 //}
207 //else // Update
208 //{
209 //    //List<IList<T>> matchedLinks = null;
210 //    if (matchedHandler != null)
211 //    {
212 //        matchedLinks = new List<IList<T>>();
213 //        Func<T, bool> handler = link =>
214 //        {
215 //            var matchedLink = Memory.GetLinkValue(link);
216 //            var matchDecision = matchedHandler(matchedLink);
217 //            if (Equals(matchDecision, Constants.Break))
218 //                return false;
219 //            if (!Equals(matchDecision, Constants.Skip))
220 //                matchedLinks.Add(matchedLink);
221 //            return true;
222 //        };
223 //        if (!Memory.Each(handler, restriction))
224 //            return Constants.Break;
225 //    }
226 //    if (!matchedLinks.IsNullOrEmpty())
227 //    {
228 //        var totalMatchedLinks = matchedLinks.Count;
229 //        for (var i = 0; i < totalMatchedLinks; i++)
230 //        {
231 //            var matchedLink = matchedLinks[i];
232 //            if (substitutedHandler != null)
233 //            {
234 //                var newValue = new List<T>(); // TODO: Prepare value to update here
235 //                // TODO: Decide is it actually needed to use Before and After
236 //                ↳ substitution handling.
237 //                var substitutedDecision = substitutedHandler(matchedLink,
238 //                ↳ newValue);
239 //                if (Equals(substitutedDecision, Constants.Break))
240 //                    return Constants.Break;
241 //                if (Equals(substitutedDecision, Constants.Continue))
242 //                {
243 //                    // Actual update here
244 //                    Memory.SetLinkValue(newValue);
245 //                }
246 //                if (Equals(substitutedDecision, Constants.Skip))
247 //                {
248 //                    // Cancel the update. TODO: decide use separate Cancel
249 //                    ↳ constant or Skip is enough?
250 //                }
251 //            }
252 //        }
253 //    }
254 //}
255 return _constants.Continue;
256 }
257
258 /// <summary>
259 /// <para>
260 /// Triggers the pattern or condition.
261 /// </para>
262 /// <para></para>
263 /// </summary>
264 /// <param name="patternOrCondition">
265 /// <para>The pattern or condition.</para>
266 /// <para></para>
267 /// </param>
268 /// <param name="matchHandler">
269 /// <para>The match handler.</para>
270 /// <para></para>
271 /// </param>
272 /// <param name="substitution">
273 /// <para>The substitution.</para>
274 /// <para></para>
275 /// </param>
276 /// <param name="substitutionHandler">
277 /// <para>The substitution handler.</para>

```

```

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, Func<IList<TLink>, TLink>
        ↳ matchHandler, IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
        ↳ substitutionHandler)
302     {
303         var constants = _constants;
304         if (patternOrCondition.IsNullOrEmpty() && substitution.IsNullOrEmpty())
305         {
306             return constants.Continue;
307         }
308         else if (patternOrCondition.EqualTo(substitution)) // Should be Each here TODO:
309             ↳ Check if it is a correct condition
310         {
311             // Or it only applies to trigger without matchHandler.
312             throw new NotImplementedException();
313         }
314         else if (!substitution.IsNullOrEmpty()) // Creation
315         {
316             var before = Array.Empty<TLink>();
317             // Что должно означать False здесь? Остановиться (перестать идти) или пропустить
318             ↳ (пройти мимо) или пустить (взять)?
319             if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
320                 ↳ constants.Break))
321             {
322                 return constants.Break;
323             }
324             var after = (IList<TLink>)substitution.ToArray();
325             if (_equalityComparer.Equals(after[0], default))
326             {
327                 var newLink = _links.Create();
328                 after[0] = newLink;
329             }
330             if (substitution.Count == 1)
331             {
332                 after = _links.GetLink(substitution[0]);
333             }
334             else if (substitution.Count == 3)
335             {
336                 //Links.Create(after);
337             }
338             else
339             {
340                 throw new NotSupportedException();
341             }
342             if (matchHandler != null)
343             {
344                 return substitutionHandler(before, after);
345             }
346             return constants.Continue;
347         }
348         else if (!patternOrCondition.IsNullOrEmpty()) // Deletion
349         {
350             if (patternOrCondition.Count == 1)

```

```

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

/// <remarks>
/// IList[IList[IList[T]]]
/// |         |         |         |
/// |         |         |-----|
/// |         |         |   link   |
/// |         |-----|
/// |         |   change   |
/// |-----|
/// |   changes
/// </remarks>

```

```

424 public IList<IList<IList<TLink>>> Trigger(IList<TLink> condition, IList<TLink>
    ↳ substitution)
425 {
426     var changes = new List<IList<IList<TLink>>>();
427     var @continue = _constants.Continue;
428     Trigger(condition, AlwaysContinue, substitution, (before, after) =>
429     {
430         var change = new[] { before, after };
431         changes.Add(change);
432         return @continue;
433     });
434     return changes;
435 }
436
437 private TLink AlwaysContinue(IList<TLink> linkToMatch) => _constants.Continue;
438 }
439 }

```

1.17 ./csharp/Platform.Data.Doublets/Doublet.cs

```

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

```



```

60 {
61     Source = source;
62     Target = target;
63 }
64
65 /// <summary>
66 /// <para>.</para>
67 /// <para>.</para>
68 /// </summary>
69 /// <returns>
70 /// <para>.</para>
71 /// <para>.</para>
72 /// </returns>
73 [MethodImpl(MethodImplOptions.AggressiveInlining)]
74 public override string ToString() => $"{Source}->{Target}";
75
76 /// <summary>
77 /// <para>.</para>
78 /// <para>.</para>
79 /// </summary>
80 /// <typeparam>
81 /// <para>.</para>
82 /// <para>.</para>
83 /// </typeparam>
84 /// <param name="other">
85 /// <para>.</para>
86 /// <para>.</para>
87 /// </param>
88 /// <returns>
89 /// <para>.</para>
90 /// <para>.</para>
91 /// </returns>
92 [MethodImpl(MethodImplOptions.AggressiveInlining)]
93 public bool Equals(Douplet<T> other) => _equalityComparer.Equals(Source, other.Source)
94     ↳ && _equalityComparer.Equals(Target, other.Target);
95
96 /// <summary>
97 /// <para>.</para>
98 /// <para>.</para>
99 /// </summary>
100 /// <typeparam>
101 /// <para>.</para>
102 /// <para>.</para>
103 /// </typeparam>
104 /// <param name="obj">
105 /// <para>.</para>
106 /// <para>.</para>
107 /// </param>
108 /// <returns>
109 /// <para>.</para>
110 /// <para>.</para>
111 /// </returns>
112 [MethodImpl(MethodImplOptions.AggressiveInlining)]
113 public override bool Equals(object obj) => obj is Douplet<T> doublet ?
114     ↳ base.Equals(doublet) : false;
115
116 /// <summary>
117 /// <para>.</para>
118 /// <para>.</para>
119 /// </summary>
120 /// <returns>
121 /// <para>.</para>
122 /// <para>.</para>
123 /// </returns>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 public override int GetHashCode() => (Source, Target).GetHashCode();
126
127 /// <summary>
128 /// <para>.</para>
129 /// <para>.</para>
130 /// </summary>
131 /// <param name="left">
132 /// <para>.</para>
133 /// <para>.</para>
134 /// </param>
135 /// <param name="right">
136 /// <para>.</para>
137 /// <para>.</para>

```

```

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 }

```

1.18 ./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 }

```

1.19 ./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 }

```

1.20 ./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.Arrays;
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
15 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
16
17 namespace Platform.Data.Doublets
18 {
19     /// <summary>
20     /// <para>
21     /// Represents the links extensions.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     public static class ILinksExtensions
26     {
27         /// <summary>
28         /// <para>
29         /// Runs the random creations using the specified links.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <typeparam name="TLink">
34         /// <para>The link.</para>
35         /// <para></para>
36         /// </typeparam>
37         /// <param name="links">
38         /// <para>The links.</para>
39         /// <para></para>
40         /// </param>
41         /// <param name="amountOfCreations">
42         /// <para>The amount of creations.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public static void RunRandomCreations<TLink>(this ILinks<TLink> links, ulong
47             ↪ amountOfCreations)

```

```

47 {
48     var random = RandomHelpers.Default;
49     var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
50     var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
51     for (var i = 0UL; i < amountOfCreations; i++)
52     {
53         var linksAddressRange = new Range<ulong>(0,
54             ↳ addressToUInt64Converter.Convert(links.Count()));
55         var source =
56             ↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
57         var target =
58             ↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
59         links.GetOrCreate(source, target);
60     }
61 }
62
63 /// <summary>
64 /// <para>
65 /// Runs the random searches using the specified links.
66 /// </para>
67 /// </summary>
68 /// <typeparam name="TLink">
69 /// <para>The link.</para>
70 /// <para></para>
71 /// </typeparam>
72 /// <param name="links">
73 /// <para>The links.</para>
74 /// <para></para>
75 /// </param>
76 /// <param name="amountOfSearches">
77 /// <para>The amount of searches.</para>
78 /// <para></para>
79 /// </param>
80 [MethodImpl(MethodImplOptions.AggressiveInlining)]
81 public static void RunRandomSearches<TLink>(this ILinks<TLink> links, ulong
82     ↳ amountOfSearches)
83 {
84     var random = RandomHelpers.Default;
85     var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
86     var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
87     for (var i = 0UL; i < amountOfSearches; i++)
88     {
89         var linksAddressRange = new Range<ulong>(0,
90             ↳ addressToUInt64Converter.Convert(links.Count()));
91         var source =
92             ↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
93         var target =
94             ↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
95         links.SearchOrDefault(source, target);
96     }
97 }
98
99 /// <summary>
100 /// <para>
101 /// Runs the random deletions using the specified links.
102 /// </para>
103 /// </summary>
104 /// <typeparam name="TLink">
105 /// <para>The link.</para>
106 /// <para></para>
107 /// </typeparam>
108 /// <param name="links">
109 /// <para>The links.</para>
110 /// <para></para>
111 /// </param>
112 /// <param name="amountOfDeletions">
113 /// <para>The amount of deletions.</para>
114 /// <para></para>
115 /// </param>
116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 public static void RunRandomDeletions<TLink>(this ILinks<TLink> links, ulong
118     ↳ amountOfDeletions)
119 {
120     var random = RandomHelpers.Default;
121     var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
122     var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;

```

```

117     var linksCount = addressToUInt64Converter.Convert(links.Count());
118     var min = amountOfDeletions > linksCount ? 0UL : linksCount - amountOfDeletions;
119     for (var i = 0UL; i < amountOfDeletions; i++)
120     {
121         linksCount = addressToUInt64Converter.Convert(links.Count());
122         if (linksCount <= min)
123         {
124             break;
125         }
126         var linksAddressRange = new Range<ulong>(min, linksCount);
127         var link =
128             ↪ uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
129         links.Delete(link);
130     }
131 }
132
133 /// <summary>
134 /// <para>
135 /// Deletes the links.
136 /// </para>
137 /// </summary>
138 /// <typeparam name="TLink">
139 /// <para>The link.</para>
140 /// <para></para>
141 /// </typeparam>
142 /// <param name="links">
143 /// <para>The links.</para>
144 /// <para></para>
145 /// </param>
146 /// <param name="linkToDelete">
147 /// <para>The link to delete.</para>
148 /// <para></para>
149 /// </param>
150 [MethodImpl(MethodImplOptions.AggressiveInlining)]
151 public static void Delete<TLink>(this ILinks<TLink> links, TLink linkToDelete) =>
152     ↪ links.Delete(new LinkAddress<TLink>(linkToDelete));
153
154 /// <remarks>
155 /// TODO: Возможно есть очень простой способ это сделать.
156 /// (Например просто удалить файл, или изменить его размер таким образом,
157 /// чтобы удалился весь контент)
158 /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
159 /// </remarks>
160 [MethodImpl(MethodImplOptions.AggressiveInlining)]
161 public static void DeleteAll<TLink>(this ILinks<TLink> links)
162 {
163     var equalityComparer = EqualityComparer<TLink>.Default;
164     var comparer = Comparer<TLink>.Default;
165     for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
166         ↪ Arithmetic.Decrement(i))
167     {
168         links.Delete(i);
169         if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
170         {
171             i = links.Count();
172         }
173     }
174 }
175
176 /// <summary>
177 /// <para>
178 /// Firsts the links.
179 /// </para>
180 /// </summary>
181 /// <typeparam name="TLink">
182 /// <para>The link.</para>
183 /// <para></para>
184 /// </typeparam>
185 /// <param name="links">
186 /// <para>The links.</para>
187 /// <para></para>
188 /// </param>
189 /// <exception cref="InvalidOperationException">
190 /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
191 /// <para></para>
192 /// </exception>

```

```

192 /// <exception cref="InvalidOperationException">
193 /// <para>В хранилище нет связей.</para>
194 /// </exception>
195 /// <returns>
196 /// <para>The first link.</para>
197 /// </returns>
198 [MethodImpl(MethodImplOptions.AggressiveInlining)]
199 public static TLink First<TLink>(this ILinks<TLink> links)
200 {
201     TLink firstLink = default;
202     var equalityComparer = EqualityComparer<TLink>.Default;
203     if (equalityComparer.Equals(links.Count(), default))
204     {
205         throw new InvalidOperationException("В хранилище нет связей.");
206     }
207     links.Each(links.Constants.Any, links.Constants.Any, link =>
208     {
209         firstLink = link[links.Constants.IndexPart];
210         return links.Constants.Break;
211     });
212     if (equalityComparer.Equals(firstLink, default))
213     {
214         throw new InvalidOperationException("В процессе поиска по хранилищу не было
215         ↳ найдено связей.");
216     }
217     return firstLink;
218 }
219
220 /// <summary>
221 /// <para>
222 /// Singles the or default using the specified links.
223 /// </para>
224 /// </summary>
225 /// <typeparam name="TLink">
226 /// <para>The link.</para>
227 /// </typeparam>
228 /// <param name="links">
229 /// <para>The links.</para>
230 /// </param>
231 /// <param name="query">
232 /// <para>The query.</para>
233 /// </param>
234 /// <returns>
235 /// <para>The result.</para>
236 /// </returns>
237 [MethodImpl(MethodImplOptions.AggressiveInlining)]
238 public static IList<TLink> SingleOrDefault<TLink>(this ILinks<TLink> links, IList<TLink>
239 ↳ query)
240 {
241     IList<TLink> result = null;
242     var count = 0;
243     var constants = links.Constants;
244     var @continue = constants.Continue;
245     var @break = constants.Break;
246     links.Each(linkHandler, query);
247     return result;
248
249     TLink linkHandler(IList<TLink> link)
250     {
251         if (count == 0)
252         {
253             result = link;
254             count++;
255             return @continue;
256         }
257         else
258         {
259             result = null;
260             return @break;
261         }
262     }
263 }
264 }
265

```

```

269 #region Paths
270
271
272 /// <remarks>
273 /// TODO: Как так? Как то что ниже может быть корректно?
274 /// Скорее всего практически не применимо
275 /// Предполагалось, что можно было конвертировать формируемый в проходе через
    → SequenceWalker
276 /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
277 /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
278 /// </remarks>
279 [MethodImpl(MethodImplOptions.AggressiveInlining)]
280 public static bool CheckPathExistance<TLink>(this ILinks<TLink> links, params TLink[]
    → path)
281 {
282     var current = path[0];
283     //EnsureLinkExists(current, "path");
284     if (!links.Exists(current))
285     {
286         return false;
287     }
288     var equalityComparer = EqualityComparer<TLink>.Default;
289     var constants = links.Constants;
290     for (var i = 1; i < path.Length; i++)
291     {
292         var next = path[i];
293         var values = links.GetLink(current);
294         var source = values[constants.SourcePart];
295         var target = values[constants.TargetPart];
296         if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
            → next))
297         {
298             //throw new InvalidOperationException(string.Format("Невозможно выбрать
            → путь, так как и Source и Target совпадают с элементом пути {0}.", next));
299             return false;
300         }
301         if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
            → target))
302         {
303             //throw new InvalidOperationException(string.Format("Невозможно продолжить
            → путь через элемент пути {0}", next));
304             return false;
305         }
306         current = next;
307     }
308     return true;
309 }
310
311 /// <remarks>
312 /// Может потребовать дополнительного стека для PathElement's при использовании
    → SequenceWalker.
313 /// </remarks>
314 [MethodImpl(MethodImplOptions.AggressiveInlining)]
315 public static TLink GetByKeyes<TLink>(this ILinks<TLink> links, TLink root, params int[]
    → path)
316 {
317     links.EnsureLinkExists(root, "root");
318     var currentLink = root;
319     for (var i = 0; i < path.Length; i++)
320     {
321         currentLink = links.GetLink(currentLink)[path[i]];
322     }
323     return currentLink;
324 }
325
326 /// <summary>
327 /// <para>
328 /// Gets the square matrix sequence element by index using the specified links.
329 /// </para>
330 /// <para></para>
331 /// </summary>
332 /// <typeparam name="TLink">
333 /// <para>The link.</para>
334 /// <para></para>
335 /// </typeparam>
336 /// <param name="links">
337 /// <para>The links.</para>
338 /// <para></para>

```

```

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

```



```

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

```

```

463 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
464     ↳ Func<IList<TLink>, TLink> handler) => links.Each(handler, links.Constants.Any,
465     ↳ source, target);
466
467 /// <summary>
468 /// <para>
469 /// Alls the links.
470 /// </para>
471 /// <para></para>
472 /// </summary>
473 /// <typeparam name="TLink">
474 /// <para>The link.</para>
475 /// <para></para>
476 /// </typeparam>
477 /// <param name="links">
478 /// <para>The links.</para>
479 /// <para></para>
480 /// </param>
481 /// <param name="restrictions">
482 /// <para>The restrictions.</para>
483 /// <para></para>
484 /// </param>
485 /// <returns>
486 /// <para>A list of i list t link</para>
487 /// <para></para>
488 /// </returns>
489 [MethodImpl(MethodImplOptions.AggressiveInlining)]
490 public static IList<IList<TLink>> All<TLink>(this ILinks<TLink> links, params TLink[]
491     ↳ restrictions)
492 {
493     var arraySize = CheckedConverter<TLink,
494         ↳ ulong>.Default.Convert(links.Count(restrictions));
495     if (arraySize > 0)
496     {
497         var array = new IList<TLink>[arraySize];
498         var filler = new ArrayFiller<IList<TLink>, TLink>(array,
499             ↳ links.Constants.Continue);
500         links.Each(filler.AddAndReturnConstant, restrictions);
501         return array;
502     }
503     else
504     {
505         return Array.Empty<IList<TLink>>();
506     }
507 }
508
509 /// <summary>
510 /// <para>
511 /// Alls the indices using the specified links.
512 /// </para>
513 /// <para></para>
514 /// </summary>
515 /// <typeparam name="TLink">
516 /// <para>The link.</para>
517 /// <para></para>
518 /// </typeparam>
519 /// <param name="links">
520 /// <para>The links.</para>
521 /// <para></para>
522 /// </param>
523 /// <param name="restrictions">
524 /// <para>The restrictions.</para>
525 /// <para></para>
526 /// </param>
527 /// <returns>
528 /// <para>A list of t link</para>
529 /// <para></para>
530 /// </returns>
531 [MethodImpl(MethodImplOptions.AggressiveInlining)]
532 public static IList<TLink> AllIndices<TLink>(this ILinks<TLink> links, params TLink[]
533     ↳ restrictions)
534 {
535     var arraySize = CheckedConverter<TLink,
536         ↳ ulong>.Default.Convert(links.Count(restrictions));
537     if (arraySize > 0)
538     {
539         var array = new TLink[arraySize];

```

```

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

```

```

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

```

```

{
    var equalityComparer = EqualityComparer<TLink>.Default;
    var any = links.Constants.Any;
    for (var i = 0; i < restrictions.Count; i++)
    {
        if (!equalityComparer.Equals(restrictions[i], any) &&
            ↪ !links.Exists(restrictions[i]))
        {
            throw new ArgumentLinkDoesNotExistsException<TLink>(restrictions[i],
                ↪ $"sequence[{i}]");
        }
    }
}

/// <summary>
/// <para>
/// Ensures the link is any or exists using the specified links.
/// </para>
/// <para></para>
/// </summary>
/// <typeparam name="TLink">
/// <para>The link.</para>
/// <para></para>
/// </typeparam>
/// <param name="links">
/// <para>The links.</para>
/// <para></para>
/// </param>
/// <param name="link">
/// <para>The link.</para>
/// <para></para>
/// </param>
/// <param name="argumentName">
/// <para>The argument name.</para>
/// <para></para>
/// </param>
/// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
/// <para></para>
/// <para></para>
/// </exception>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links, TLink link,
    ↪ string argumentName)
{
    var equalityComparer = EqualityComparer<TLink>.Default;
    if (!equalityComparer.Equals(link, links.Constants.Any) && !links.Exists(link))
    {
        throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
    }
}

/// <summary>
/// <para>
/// Ensures the link is itself or exists using the specified links.
/// </para>
/// <para></para>
/// </summary>
/// <typeparam name="TLink">
/// <para>The link.</para>
/// <para></para>
/// </typeparam>
/// <param name="links">
/// <para>The links.</para>
/// <para></para>
/// </param>
/// <param name="link">
/// <para>The link.</para>
/// <para></para>
/// </param>
/// <param name="argumentName">
/// <para>The argument name.</para>
/// <para></para>
/// </param>
/// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
/// <para></para>
/// <para></para>
/// </exception>
[MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

755 public static void EnsureLinkIsItselfOrExists<TLink>(this ILinks<TLink> links, TLink
    ↳ link, string argumentName)
756 {
757     var equalityComparer = EqualityComparer<TLink>.Default;
758     if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))
759     {
760         throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
761     }
762 }
763
764 /// <param name="links">Хранилище связей.</param>
765 [MethodImpl(MethodImplOptions.AggressiveInlining)]
766 public static void EnsureDoesNotExists<TLink>(this ILinks<TLink> links, TLink source,
    ↳ TLink target)
767 {
768     if (links.Exists(source, target))
769     {
770         throw new LinkWithSameValueAlreadyExistsException();
771     }
772 }
773
774 /// <param name="links">Хранилище связей.</param>
775 [MethodImpl(MethodImplOptions.AggressiveInlining)]
776 public static void EnsureNoUsages<TLink>(this ILinks<TLink> links, TLink link)
777 {
778     if (links.HasUsages(link))
779     {
780         throw new ArgumentLinkHasDependenciesException<TLink>(link);
781     }
782 }
783
784 /// <param name="links">Хранилище связей.</param>
785 [MethodImpl(MethodImplOptions.AggressiveInlining)]
786 public static void EnsureCreated<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ addresses) => links.EnsureCreated(links.Create, addresses);
787
788 /// <param name="links">Хранилище связей.</param>
789 [MethodImpl(MethodImplOptions.AggressiveInlining)]
790 public static void EnsurePointsCreated<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ addresses) => links.EnsureCreated(links.CreatePoint, addresses);
791
792 /// <param name="links">Хранилище связей.</param>
793 [MethodImpl(MethodImplOptions.AggressiveInlining)]
794 public static void EnsureCreated<TLink>(this ILinks<TLink> links, Func<TLink> creator,
    ↳ params TLink[] addresses)
795 {
796     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
797     var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
798     var nonExistentAddresses = new HashSet<TLink>(addresses.Where(x =>
    ↳ !links.Exists(x)));
799     if (nonExistentAddresses.Count > 0)
800     {
801         var max = nonExistentAddresses.Max();
802         max = uInt64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
    ↳ Convert(max),
    ↳ addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
    ↳ imum)));
803         var createdLinks = new List<TLink>();
804         var equalityComparer = EqualityComparer<TLink>.Default;
805         TLink createdLink = creator();
806         while (!equalityComparer.Equals(createdLink, max))
807         {
808             createdLinks.Add(createdLink);
809         }
810         for (var i = 0; i < createdLinks.Count; i++)
811         {
812             if (!nonExistentAddresses.Contains(createdLinks[i]))
813             {
814                 links.Delete(createdLinks[i]);
815             }
816         }
817     }
818 }
819
820 #endregion
821
822 /// <param name="links">Хранилище связей.</param>
823 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

824 public static TLink CountUsages<TLink>(this ILinks<TLink> links, TLink link)
825 {
826     var constants = links.Constants;
827     var values = links.GetLink(link);
828     TLink usagesAsSource = links.Count(new Link<TLink>(constants.Any, link,
829         ↪ constants.Any));
830     var equalityComparer = EqualityComparer<TLink>.Default;
831     if (equalityComparer.Equals(values[constants.SourcePart], link))
832     {
833         usagesAsSource = Arithmetic<TLink>.Decrement(usagesAsSource);
834     }
835     TLink usagesAsTarget = links.Count(new Link<TLink>(constants.Any, constants.Any,
836         ↪ link));
837     if (equalityComparer.Equals(values[constants.TargetPart], link))
838     {
839         usagesAsTarget = Arithmetic<TLink>.Decrement(usagesAsTarget);
840     }
841     return Arithmetic<TLink>.Add(usagesAsSource, usagesAsTarget);
842 }
843
844 /// <param name="links">Хранилище связей.</param>
845 [MethodImpl(MethodImplOptions.AggressiveInlining)]
846 public static bool HasUsages<TLink>(this ILinks<TLink> links, TLink link) =>
847     ↪ Comparer<TLink>.Default.Compare(links.CountUsages(link), default) > 0;
848
849 /// <param name="links">Хранилище связей.</param>
850 [MethodImpl(MethodImplOptions.AggressiveInlining)]
851 public static bool Equals<TLink>(this ILinks<TLink> links, TLink link, TLink source,
852     ↪ TLink target)
853 {
854     var constants = links.Constants;
855     var values = links.GetLink(link);
856     var equalityComparer = EqualityComparer<TLink>.Default;
857     return equalityComparer.Equals(values[constants.SourcePart], source) &&
858         ↪ equalityComparer.Equals(values[constants.TargetPart], target);
859 }
860
861 /// <summary>
862 /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
863 /// </summary>
864 /// <param name="links">Хранилище связей.</param>
865 /// <param name="source">Индекс связи, которая является началом для искомой
866     ↪ связи.</param>
867 /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
868 /// <returns>Индекс искомой связи с указанными Source (началом) и Target
869     ↪ (концом).</returns>
870 [MethodImpl(MethodImplOptions.AggressiveInlining)]
871 public static TLink SearchOrDefault<TLink>(this ILinks<TLink> links, TLink source, TLink
872     ↪ target)
873 {
874     var constants = links.Constants;
875     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break, default);
876     links.Each(setter.SetFirstAndReturnFalse, constants.Any, source, target);
877     return setter.Result;
878 }
879
880 /// <param name="links">Хранилище связей.</param>
881 [MethodImpl(MethodImplOptions.AggressiveInlining)]
882 public static TLink Create<TLink>(this ILinks<TLink> links) => links.Create(null);
883
884 /// <param name="links">Хранилище связей.</param>
885 [MethodImpl(MethodImplOptions.AggressiveInlining)]
886 public static TLink CreatePoint<TLink>(this ILinks<TLink> links)
887 {
888     var link = links.Create();
889     return links.Update(link, link, link);
890 }
891
892 /// <param name="links">Хранилище связей.</param>
893 [MethodImpl(MethodImplOptions.AggressiveInlining)]
894 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
895     ↪ target) => links.Update(links.Create(), source, target);
896
897 /// <summary>
898 /// Обновляет связь с указанными началом (Source) и концом (Target)
899 /// на связь с указанными началом (NewSource) и концом (NewTarget).
900 /// </summary>
901 /// <param name="links">Хранилище связей.</param>

```

```

893 /// <param name="link">Индекс обновляемой связи.</param>
894 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
    ↳ выполняется обновление.</param>
895 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
    ↳ выполняется обновление.</param>
896 /// <returns>Индекс обновлённой связи.</returns>
897 [MethodImpl(MethodImplOptions.AggressiveInlining)]
898 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));
899
900 /// <summary>
901 /// Обновляет связь с указанными началом (Source) и концом (Target)
902 /// на связь с указанными началом (NewSource) и концом (NewTarget).
903 /// </summary>
904 /// <param name="links">Хранилище связей.</param>
905 /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
    ↳ может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
    ↳ Itself - требование установить ссылку на себя, 1..∞ конкретный адрес другой
    ↳ связи.</param>
906 /// <returns>Индекс обновлённой связи.</returns>
907 [MethodImpl(MethodImplOptions.AggressiveInlining)]
908 public static TLink Update<TLink>(this ILinks<TLink> links, params TLink[] restrictions)
909 {
910     if (restrictions.Length == 2)
911     {
912         return links.MergeAndDelete(restrictions[0], restrictions[1]);
913     }
914     if (restrictions.Length == 4)
915     {
916         return links.UpdateOrCreateOrGet(restrictions[0], restrictions[1],
            ↳ restrictions[2], restrictions[3]);
917     }
918     else
919     {
920         return links.Update(new LinkAddress<TLink>(restrictions[0]), restrictions);
921     }
922 }
923
924 /// <summary>
925 /// <para>
926 /// Resolves the constant as self reference using the specified links.
927 /// </para>
928 /// <para></para>
929 /// </summary>
930 /// <typeparam name="TLink">
931 /// <para>The link.</para>
932 /// <para></para>
933 /// </typeparam>
934 /// <param name="links">
935 /// <para>The links.</para>
936 /// <para></para>
937 /// </param>
938 /// <param name="constant">
939 /// <para>The constant.</para>
940 /// <para></para>
941 /// </param>
942 /// <param name="restrictions">
943 /// <para>The restrictions.</para>
944 /// <para></para>
945 /// </param>
946 /// <param name="substitution">
947 /// <para>The substitution.</para>
948 /// <para></para>
949 /// </param>
950 /// <returns>
951 /// <para>A list of t link</para>
952 /// <para></para>
953 /// </returns>
954 [MethodImpl(MethodImplOptions.AggressiveInlining)]
955 public static IList<TLink> ResolveConstantAsSelfReference<TLink>(this ILinks<TLink>
    ↳ links, TLink constant, IList<TLink> restrictions, IList<TLink> substitution)
956 {
957     var equalityComparer = EqualityComparer<TLink>.Default;
958     var constants = links.Constants;
959     var restrictionsIndex = restrictions[constants.IndexPart];
960     var substitutionIndex = substitution[constants.IndexPart];

```



```

961     if (equalityComparer.Equals(substitutionIndex, default))
962     {
963         substitutionIndex = restrictionsIndex;
964     }
965     var source = substitution[constants.SourcePart];
966     var target = substitution[constants.TargetPart];
967     source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;
968     target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
969     return new Link<TLink>(substitutionIndex, source, target);
970 }
971
972 /// <summary>
973 /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
974   ↳ с указанными Source (началом) и Target (концом).
975 /// </summary>
976 /// <param name="links">Хранилище связей.</param>
977 /// <param name="source">Индекс связи, которая является началом на создаваемой
978   ↳ связи.</param>
979 /// <param name="target">Индекс связи, которая является концом для создаваемой
980   ↳ связи.</param>
981 /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
982 [MethodImpl(MethodImplOptions.AggressiveInlining)]
983 public static TLink GetOrCreate<TLink>(this ILinks<TLink> links, TLink source, TLink
984   ↳ target)
985 {
986     var link = links.SearchOrDefault(source, target);
987     if (EqualityComparer<TLink>.Default.Equals(link, default))
988     {
989         link = links.CreateAndUpdate(source, target);
990     }
991     return link;
992 }
993
994 /// <summary>
995 /// Обновляет связь с указанными началом (Source) и концом (Target)
996   ↳ на связь с указанными началом (NewSource) и концом (NewTarget).
997 /// </summary>
998 /// <param name="links">Хранилище связей.</param>
999 /// <param name="source">Индекс связи, которая является началом обновляемой
1000   ↳ связи.</param>
1001 /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
1002 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
1003   ↳ выполняется обновление.</param>
1004 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
1005   ↳ выполняется обновление.</param>
1006 /// <returns>Индекс обновлённой связи.</returns>
1007 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1008 public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
1009   ↳ TLink target, TLink newSource, TLink newTarget)
1010 {
1011     var equalityComparer = EqualityComparer<TLink>.Default;
1012     var link = links.SearchOrDefault(source, target);
1013     if (equalityComparer.Equals(link, default))
1014     {
1015         return links.CreateAndUpdate(newSource, newTarget);
1016     }
1017     if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
1018   ↳ target))
1019     {
1020         return link;
1021     }
1022     return links.Update(link, newSource, newTarget);
1023 }
1024
1025 /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1026 /// <param name="links">Хранилище связей.</param>
1027 /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
1028 /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
1029 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1030 public static TLink DeleteIfExists<TLink>(this ILinks<TLink> links, TLink source, TLink
1031   ↳ target)
1032 {
1033     var link = links.SearchOrDefault(source, target);
1034     if (!EqualityComparer<TLink>.Default.Equals(link, default))
1035     {
1036         links.Delete(link);
1037         return link;
1038     }

```

```

1028     }
1029     return default;
1030 }
1031
1032 /// <summary>Удаляет несколько связей.</summary>
1033 /// <param name="links">Хранилище связей.</param>
1034 /// <param name="deletedLinks">Список адресов связей к удалению.</param>
1035 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1036 public static void DeleteMany<TLink>(this ILinks<TLink> links, IList<TLink> deletedLinks)
1037 {
1038     for (int i = 0; i < deletedLinks.Count; i++)
1039     {
1040         links.Delete(deletedLinks[i]);
1041     }
1042 }
1043
1044 /// <remarks>Before execution of this method ensure that deleted link is detached (all
1045 ↪ values - source and target are reset to null) or it might enter into infinite
1046 ↪ recursion.</remarks>
1047 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1048 public static void DeleteAllUsages<TLink>(this ILinks<TLink> links, TLink linkIndex)
1049 {
1050     var anyConstant = links.Constants.Any;
1051     var usagesAsSourceQuery = new Link<TLink>(anyConstant, linkIndex, anyConstant);
1052     links.DeleteByQuery(usagesAsSourceQuery);
1053     var usagesAsTargetQuery = new Link<TLink>(anyConstant, linkIndex, anyConstant);
1054     links.DeleteByQuery(usagesAsTargetQuery);
1055 }
1056
1057 /// <summary>
1058 /// <para>
1059 /// Deletes the by query using the specified links.
1060 /// </para>
1061 /// </summary>
1062 /// <typeparam name="TLink">
1063 /// <para>The link.</para>
1064 /// </typeparam>
1065 /// <param name="links">
1066 /// <para>The links.</para>
1067 /// </param>
1068 /// <param name="query">
1069 /// <para>The query.</para>
1070 /// </param>
1071 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1072 public static void DeleteByQuery<TLink>(this ILinks<TLink> links, Link<TLink> query)
1073 {
1074     var count = CheckedConverter<TLink, long>.Default.Convert(links.Count(query));
1075     if (count > 0)
1076     {
1077         var queryResult = new TLink[count];
1078         var queryResultFiller = new ArrayFiller<TLink, TLink>(queryResult,
1079             ↪ links.Constants.Continue);
1080         links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
1081         for (var i = count - 1; i >= 0; i--)
1082         {
1083             links.Delete(queryResult[i]);
1084         }
1085     }
1086 }
1087
1088 }
1089
1090 // TODO: Move to Platform.Data
1091 /// <summary>
1092 /// <para>
1093 /// Determines whether are values reset.
1094 /// </para>
1095 /// </summary>
1096 /// <typeparam name="TLink">
1097 /// <para>The link.</para>
1098 /// </typeparam>
1099 /// <param name="links">
1100 /// <para>The links.</para>
1101 /// </param>
1102

```

```

1103     /// </param>
1104     /// <param name="linkIndex">
1105     /// <para>The link index.</para>
1106     /// <para></para>
1107     /// </param>
1108     /// <returns>
1109     /// <para>The bool</para>
1110     /// <para></para>
1111     /// </returns>
1112     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1113     public static bool AreValuesReset<TLink>(this ILinks<TLink> links, TLink linkIndex)
1114     {
1115         var nullConstant = links.Constants.Null;
1116         var equalityComparer = EqualityComparer<TLink>.Default;
1117         var link = links.GetLink(linkIndex);
1118         for (int i = 1; i < link.Count; i++)
1119         {
1120             if (!equalityComparer.Equals(link[i], nullConstant))
1121             {
1122                 return false;
1123             }
1124         }
1125         return true;
1126     }
1127
1128     // TODO: Create a universal version of this method in Platform.Data (with using of for
1129     // ↳ loop)
1130     /// <summary>
1131     /// <para>
1132     /// Resets the values using the specified links.
1133     /// </para>
1134     /// <para></para>
1135     /// </summary>
1136     /// <typeparam name="TLink">
1137     /// <para>The link.</para>
1138     /// <para></para>
1139     /// </typeparam>
1140     /// <param name="links">
1141     /// <para>The links.</para>
1142     /// <para></para>
1143     /// </param>
1144     /// <param name="linkIndex">
1145     /// <para>The link index.</para>
1146     /// <para></para>
1147     /// </param>
1148     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1149     public static void ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1150     {
1151         var nullConstant = links.Constants.Null;
1152         var updateRequest = new Link<TLink>(linkIndex, nullConstant, nullConstant);
1153         links.Update(updateRequest);
1154     }
1155
1156     // TODO: Create a universal version of this method in Platform.Data (with using of for
1157     // ↳ loop)
1158     /// <summary>
1159     /// <para>
1160     /// Enforces the reset values using the specified links.
1161     /// </para>
1162     /// <para></para>
1163     /// </summary>
1164     /// <typeparam name="TLink">
1165     /// <para>The link.</para>
1166     /// <para></para>
1167     /// </typeparam>
1168     /// <param name="links">
1169     /// <para>The links.</para>
1170     /// <para></para>
1171     /// </param>
1172     /// <param name="linkIndex">
1173     /// <para>The link index.</para>
1174     /// <para></para>
1175     /// </param>
1176     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1177     public static void EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1178     {
1179         if (!links.AreValuesReset(linkIndex))
1180         {

```

```

1179         links.ResetValues(linkIndex);
1180     }
1181 }
1182
1183 /// <summary>
1184 /// Merging two usages graphs, all children of old link moved to be children of new link
1185   ↳ or deleted.
1186 /// </summary>
1187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1188 public static TLink MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1189   ↳ TLink newLinkIndex)
1190 {
1191     var addressToInt64Converter = CheckedConverter<TLink, long>.Default;
1192     var equalityComparer = EqualityComparer<TLink>.Default;
1193     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1194     {
1195         var constants = links.Constants;
1196         var usagesAsSourceQuery = new Link<TLink>(constants.Any, oldLinkIndex,
1197   ↳ constants.Any);
1198         var usagesAsSourceCount =
1199   ↳ addressToInt64Converter.Convert(links.Count(usagesAsSourceQuery));
1200         var usagesAsTargetQuery = new Link<TLink>(constants.Any, constants.Any,
1201   ↳ oldLinkIndex);
1202         var usagesAsTargetCount =
1203   ↳ addressToInt64Converter.Convert(links.Count(usagesAsTargetQuery));
1204         var isStandalonePoint = Point<TLink>.IsFullPoint(links.GetLink(oldLinkIndex)) &&
1205   ↳ usagesAsSourceCount == 1 && usagesAsTargetCount == 1;
1206         if (!isStandalonePoint)
1207         {
1208             var totalUsages = usagesAsSourceCount + usagesAsTargetCount;
1209             if (totalUsages > 0)
1210             {
1211                 var usages = ArrayPool.Allocate<TLink>(totalUsages);
1212                 var usagesFiller = new ArrayFiller<TLink, TLink>(usages,
1213   ↳ links.Constants.Continue);
1214                 var i = 0L;
1215                 if (usagesAsSourceCount > 0)
1216                 {
1217                     links.Each(usagesFiller.AddFirstAndReturnConstant,
1218   ↳ usagesAsSourceQuery);
1219                     for (; i < usagesAsSourceCount; i++)
1220                     {
1221                         var usage = usages[i];
1222                         if (!equalityComparer.Equals(usage, oldLinkIndex))
1223                         {
1224                             links.Update(usage, newLinkIndex, links.GetTarget(usage));
1225                         }
1226                     }
1227                 }
1228                 if (usagesAsTargetCount > 0)
1229                 {
1230                     links.Each(usagesFiller.AddFirstAndReturnConstant,
1231   ↳ usagesAsTargetQuery);
1232                     for (; i < usages.Length; i++)
1233                     {
1234                         var usage = usages[i];
1235                         if (!equalityComparer.Equals(usage, oldLinkIndex))
1236                         {
1237                             links.Update(usage, links.GetSource(usage), newLinkIndex);
1238                         }
1239                     }
1240                 }
1241                 ArrayPool.Free(usages);
1242             }
1243         }
1244     }
1245     return newLinkIndex;
1246 }
1247
1248 /// <summary>
1249 /// Replace one link with another (replaced link is deleted, children are updated or
1250   ↳ deleted).
1251 /// </summary>
1252 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1253 public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1254   ↳ TLink newLinkIndex)
1255 {
1256     var equalityComparer = EqualityComparer<TLink>.Default;

```

```

1245         if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1246         {
1247             links.MergeUsages(oldLinkIndex, newLinkIndex);
1248             links.Delete(oldLinkIndex);
1249         }
1250         return newLinkIndex;
1251     }
1252
1253     /// <summary>
1254     /// <para>
1255     /// Decorates the with automatic uniqueness and usages resolution using the specified
1256     ↪ links.
1257     /// </para>
1258     /// <para></para>
1259     /// </summary>
1260     /// <typeparam name="TLink">
1261     /// <para>The link.</para>
1262     /// <para></para>
1263     /// </typeparam>
1264     /// <param name="links">
1265     /// <para>The links.</para>
1266     /// <para></para>
1267     /// </param>
1268     /// <returns>
1269     /// <para>The links.</para>
1270     /// <para></para>
1271     /// </returns>
1272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1273     public static ILinks<TLink>
1274     ↪ DecorateWithAutomaticUniquenessAndUsagesResolution<TLink>(this ILinks<TLink> links)
1275     {
1276         links = new LinksCascadeUsagesResolver<TLink>(links);
1277         links = new NonNullContentsLinkDeletionResolver<TLink>(links);
1278         links = new LinksCascadeUniquenessAndUsagesResolver<TLink>(links);
1279         return links;
1280     }
1281
1282     /// <summary>
1283     /// <para>
1284     /// Formats the links.
1285     /// </para>
1286     /// <para></para>
1287     /// </summary>
1288     /// <typeparam name="TLink">
1289     /// <para>The link.</para>
1290     /// <para></para>
1291     /// </typeparam>
1292     /// <param name="links">
1293     /// <para>The links.</para>
1294     /// <para></para>
1295     /// </param>
1296     /// <param name="link">
1297     /// <para>The link.</para>
1298     /// <para></para>
1299     /// </param>
1300     /// <returns>
1301     /// <para>The string</para>
1302     /// <para></para>
1303     /// </returns>
1304     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1305     public static string Format<TLink>(this ILinks<TLink> links, IList<TLink> link)
1306     {
1307         var constants = links.Constants;
1308         return $"{(link[constants.IndexPart]): {link[constants.SourcePart]}}
1309         ↪ {link[constants.TargetPart]}";
1310     }
1311
1312     /// <summary>
1313     /// <para>
1314     /// Formats the links.
1315     /// </para>
1316     /// <para></para>
1317     /// </summary>
1318     /// <typeparam name="TLink">
1319     /// <para>The link.</para>
1320     /// <para></para>
1321     /// </typeparam>
1322     /// <param name="links">

```

```

1320     /// <para>The links.</para>
1321     /// <para></para>
1322     /// </param>
1323     /// <param name="link">
1324     /// <para>The link.</para>
1325     /// <para></para>
1326     /// </param>
1327     /// <returns>
1328     /// <para>The string</para>
1329     /// <para></para>
1330     /// </returns>
1331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1332     public static string Format<TLink>(this ILinks<TLink> links, TLink link) =>
        ↪ links.Format(links.GetLink(link));
1333 }
1334 }

```

1.21 ./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>,
        ↪ LinksConstants<TLink>>, ILinks<TLink>
14     {
15     }
16 }

```

1.22 ./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
27         private static readonly LinksConstants<TLink> _constants =
            ↪ Default<LinksConstants<TLink>>.Instance;
28         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
29
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
        ↪ Target);
66
67     /// <summary>
68     /// <para>
69     /// Initializes a new <see cref="Link"/> instance.
70     /// </para>
71     /// <para></para>
72     /// </summary>
73     /// <param name="values">
74     /// <para>A values.</para>
75     /// <para></para>
76     /// </param>
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     public Link(ICollection<TLink> values) => SetValues(values, out Index, out Source, out Target);
79
80     /// <summary>
81     /// <para>
82     /// Initializes a new <see cref="Link"/> instance.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <param name="other">
87     /// <para>A other.</para>
88     /// <para></para>
89     /// </param>
90     /// <exception cref="NotSupportedException">
91     /// <para></para>
92     /// <para></para>
93     /// </exception>
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     public Link(object other)
96     {
97         if (other is Link<TLink> otherLink)
98         {
99             SetValues(ref otherLink, out Index, out Source, out Target);
100         }
101         else if (other is ICollection<TLink> otherList)
102         {
103             SetValues(otherList, out Index, out Source, out Target);
104         }
105         else
106         {
107             throw new NotSupportedException();
108         }
109     }
110
111     /// <summary>
112     /// <para>
113     /// Initializes a new <see cref="Link"/> instance.
114     /// </para>
115     /// <para></para>
116     /// </summary>
117     /// <param name="other">

```

```

118     /// <para>A other.</para>
119     /// <para></para>
120     /// </param>
121     [MethodImpl(MethodImplOptions.AggressiveInlining)]
122     public Link(ref Link<TLink> other) => SetValues(ref other, out Index, out Source, out
        ↪ Target);
123
124     /// <summary>
125     /// <para>
126     /// Initializes a new <see cref="Link"/> instance.
127     /// </para>
128     /// <para></para>
129     /// </summary>
130     /// <param name="index">
131     /// <para>A index.</para>
132     /// <para></para>
133     /// </param>
134     /// <param name="source">
135     /// <para>A source.</para>
136     /// <para></para>
137     /// </param>
138     /// <param name="target">
139     /// <para>A target.</para>
140     /// <para></para>
141     /// </param>
142     [MethodImpl(MethodImplOptions.AggressiveInlining)]
143     public Link(TLink index, TLink source, TLink target)
144     {
145         Index = index;
146         Source = source;
147         Target = target;
148     }
149
150     /// <summary>
151     /// <para>
152     /// Sets the values using the specified other.
153     /// </para>
154     /// <para></para>
155     /// </summary>
156     /// <param name="other">
157     /// <para>The other.</para>
158     /// <para></para>
159     /// </param>
160     /// <param name="index">
161     /// <para>The index.</para>
162     /// <para></para>
163     /// </param>
164     /// <param name="source">
165     /// <para>The source.</para>
166     /// <para></para>
167     /// </param>
168     /// <param name="target">
169     /// <para>The target.</para>
170     /// <para></para>
171     /// </param>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     private static void SetValues(ref Link<TLink> other, out TLink index, out TLink source,
        ↪ out TLink target)
174     {
175         index = other.Index;
176         source = other.Source;
177         target = other.Target;
178     }
179
180     /// <summary>
181     /// <para>
182     /// Sets the values using the specified values.
183     /// </para>
184     /// <para></para>
185     /// </summary>
186     /// <param name="values">
187     /// <para>The values.</para>
188     /// <para></para>
189     /// </param>
190     /// <param name="index">
191     /// <para>The index.</para>
192     /// <para></para>
193     /// </param>

```



```

194     /// <param name="source">
195     /// <para>The source.</para>
196     /// <para></para>
197     /// </param>
198     /// <param name="target">
199     /// <para>The target.</para>
200     /// <para></para>
201     /// </param>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     private static void SetValues(ICollection<TLink> values, out TLink index, out TLink source,
204     ↪ out TLink target)
205     {
206         switch (values.Count)
207         {
208             case 3:
209                 index = values[0];
210                 source = values[1];
211                 target = values[2];
212                 break;
213             case 2:
214                 index = values[0];
215                 source = values[1];
216                 target = default;
217                 break;
218             case 1:
219                 index = values[0];
220                 source = default;
221                 target = default;
222                 break;
223             default:
224                 index = default;
225                 source = default;
226                 target = default;
227                 break;
228         }
229     }
230
231     /// <summary>
232     /// <para>
233     /// Gets the hash code.
234     /// </para>
235     /// </summary>
236     /// <returns>
237     /// <para>The int</para>
238     /// <para></para>
239     /// </returns>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     public override int GetHashCode() => (Index, Source, Target).GetHashCode();
242
243     /// <summary>
244     /// <para>
245     /// Determines whether this instance is null.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     /// <returns>
250     /// <para>The bool</para>
251     /// <para></para>
252     /// </returns>
253     [MethodImpl(MethodImplOptions.AggressiveInlining)]
254     public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
255     && _equalityComparer.Equals(Source, _constants.Null)
256     && _equalityComparer.Equals(Target, _constants.Null);
257
258     /// <summary>
259     /// <para>
260     /// Determines whether this instance equals.
261     /// </para>
262     /// <para></para>
263     /// </summary>
264     /// <param name="other">
265     /// <para>The other.</para>
266     /// <para></para>
267     /// </param>
268     /// <returns>
269     /// <para>The bool</para>
270     /// <para></para>
271     /// </returns>

```

```

272 [MethodImpl(MethodImplOptions.AggressiveInlining)]
273 public override bool Equals(object other) => other is Link<TLink> &&
    ↳ Equals((Link<TLink>)other);
274
275 /// <summary>
276 /// <para>
277 /// Determines whether this instance equals.
278 /// </para>
279 /// <para></para>
280 /// </summary>
281 /// <param name="other">
282 /// <para>The other.</para>
283 /// <para></para>
284 /// </param>
285 /// <returns>
286 /// <para>The bool</para>
287 /// <para></para>
288 /// </returns>
289 [MethodImpl(MethodImplOptions.AggressiveInlining)]
290 public bool Equals(Link<TLink> other) => _equalityComparer.Equals(Index, other.Index)
291                                     && _equalityComparer.Equals(Source, other.Source)
292                                     && _equalityComparer.Equals(Target, other.Target);
293
294 /// <summary>
295 /// <para>
296 /// Returns the string using the specified index.
297 /// </para>
298 /// <para></para>
299 /// </summary>
300 /// <param name="index">
301 /// <para>The index.</para>
302 /// <para></para>
303 /// </param>
304 /// <param name="source">
305 /// <para>The source.</para>
306 /// <para></para>
307 /// </param>
308 /// <param name="target">
309 /// <para>The target.</para>
310 /// <para></para>
311 /// </param>
312 /// <returns>
313 /// <para>The string</para>
314 /// <para></para>
315 /// </returns>
316 [MethodImpl(MethodImplOptions.AggressiveInlining)]
317 public static string ToString(TLink index, TLink source, TLink target) => $"{index}:
    ↳ {source}->{target}";
318
319 /// <summary>
320 /// <para>
321 /// Returns the string using the specified source.
322 /// </para>
323 /// <para></para>
324 /// </summary>
325 /// <param name="source">
326 /// <para>The source.</para>
327 /// <para></para>
328 /// </param>
329 /// <param name="target">
330 /// <para>The target.</para>
331 /// <para></para>
332 /// </param>
333 /// <returns>
334 /// <para>The string</para>
335 /// <para></para>
336 /// </returns>
337 [MethodImpl(MethodImplOptions.AggressiveInlining)]
338 public static string ToString(TLink source, TLink target) => $"{source}->{target}";
339
340 [MethodImpl(MethodImplOptions.AggressiveInlining)]
341 public static implicit operator TLink[] (Link<TLink> link) => link.ToArray();
342
343 [MethodImpl(MethodImplOptions.AggressiveInlining)]
344 public static implicit operator Link<TLink> (TLink[] linkArray) => new
    ↳ Link<TLink>(linkArray);
345
346 /// <summary>

```

```

347     /// <para>
348     /// Returns the string.
349     /// </para>
350     /// <para></para>
351     /// </summary>
352     /// <returns>
353     /// <para>The string</para>
354     /// <para></para>
355     /// </returns>
356     [MethodImpl(MethodImplOptions.AggressiveInlining)]
357     public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
        ↳ ToString(Source, Target) : ToString(Index, Source, Target);
358
359     #region IList
360
361     /// <summary>
362     /// <para>
363     /// Gets the count value.
364     /// </para>
365     /// <para></para>
366     /// </summary>
367     public int Count
368     {
369         [MethodImpl(MethodImplOptions.AggressiveInlining)]
370         get => Length;
371     }
372
373     /// <summary>
374     /// <para>
375     /// Gets the is read only value.
376     /// </para>
377     /// <para></para>
378     /// </summary>
379     public bool IsReadOnly
380     {
381         [MethodImpl(MethodImplOptions.AggressiveInlining)]
382         get => true;
383     }
384
385     /// <summary>
386     /// <para>
387     /// The not supported exception.
388     /// </para>
389     /// <para></para>
390     /// </summary>
391     public TLink this[int index]
392     {
393         [MethodImpl(MethodImplOptions.AggressiveInlining)]
394         get
395         {
396             Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
397                 ↳ nameof(index));
398             if (index == _constants.IndexPart)
399             {
400                 return Index;
401             }
402             if (index == _constants.SourcePart)
403             {
404                 return Source;
405             }
406             if (index == _constants.TargetPart)
407             {
408                 return Target;
409             }
410             throw new NotSupportedException(); // Impossible path due to
411                 ↳ Ensure.ArgumentInRange
412         }
413         [MethodImpl(MethodImplOptions.AggressiveInlining)]
414         set => throw new NotSupportedException();
415     }
416
417     /// <summary>
418     /// <para>
419     /// Gets the enumerator.
420     /// </para>
421     /// <para></para>
422     /// </summary>
423     /// <returns>
424     /// <para>The enumerator</para>

```

```

423     /// <para></para>
424     /// </returns>
425     [MethodImpl(MethodImplOptions.AggressiveInlining)]
426     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
427
428     /// <summary>
429     /// <para>
430     /// Gets the enumerator.
431     /// </para>
432     /// <para></para>
433     /// </summary>
434     /// <returns>
435     /// <para>An enumerator of t link</para>
436     /// <para></para>
437     /// </returns>
438     [MethodImpl(MethodImplOptions.AggressiveInlining)]
439     public IEnumerator<TLink> GetEnumerator()
440     {
441         yield return Index;
442         yield return Source;
443         yield return Target;
444     }
445
446     /// <summary>
447     /// <para>
448     /// Adds the item.
449     /// </para>
450     /// <para></para>
451     /// </summary>
452     /// <param name="item">
453     /// <para>The item.</para>
454     /// <para></para>
455     /// </param>
456     [MethodImpl(MethodImplOptions.AggressiveInlining)]
457     public void Add(TLink item) => throw new NotSupportedException();
458
459     /// <summary>
460     /// <para>
461     /// Clears this instance.
462     /// </para>
463     /// <para></para>
464     /// </summary>
465     [MethodImpl(MethodImplOptions.AggressiveInlining)]
466     public void Clear() => throw new NotSupportedException();
467
468     /// <summary>
469     /// <para>
470     /// Determines whether this instance contains.
471     /// </para>
472     /// <para></para>
473     /// </summary>
474     /// <param name="item">
475     /// <para>The item.</para>
476     /// <para></para>
477     /// </param>
478     /// <returns>
479     /// <para>The bool</para>
480     /// <para></para>
481     /// </returns>
482     [MethodImpl(MethodImplOptions.AggressiveInlining)]
483     public bool Contains(TLink item) => IndexOf(item) >= 0;
484
485     /// <summary>
486     /// <para>
487     /// Copies the to using the specified array.
488     /// </para>
489     /// <para></para>
490     /// </summary>
491     /// <param name="array">
492     /// <para>The array.</para>
493     /// <para></para>
494     /// </param>
495     /// <param name="arrayIndex">
496     /// <para>The array index.</para>
497     /// <para></para>
498     /// </param>
499     /// <exception cref="InvalidOperationException">
500     /// <para></para>

```

```

501 /// <para></para>
502 /// </exception>
503 [MethodImpl(MethodImplOptions.AggressiveInlining)]
504 public void CopyTo(TLink[] array, int arrayIndex)
505 {
506     Ensure.OnDebug.ArgumentNotNull(array, nameof(array));
507     Ensure.OnDebug.ArgumentInRange(arrayIndex, new Range<int>(0, array.Length - 1),
508         ↪ nameof(arrayIndex));
509     if (arrayIndex + Length > array.Length)
510     {
511         throw new InvalidOperationException();
512     }
513     array[arrayIndex++] = Index;
514     array[arrayIndex++] = Source;
515     array[arrayIndex] = Target;
516 }
517 /// <summary>
518 /// <para>
519 /// Determines whether this instance remove.
520 /// </para>
521 /// <para></para>
522 /// </summary>
523 /// <param name="item">
524 /// <para>The item.</para>
525 /// <para></para>
526 /// </param>
527 /// <returns>
528 /// <para>The bool</para>
529 /// <para></para>
530 /// </returns>
531 [MethodImpl(MethodImplOptions.AggressiveInlining)]
532 public bool Remove(TLink item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
533
534 /// <summary>
535 /// <para>
536 /// Indexes the of using the specified item.
537 /// </para>
538 /// <para></para>
539 /// </summary>
540 /// <param name="item">
541 /// <para>The item.</para>
542 /// <para></para>
543 /// </param>
544 /// <returns>
545 /// <para>The int</para>
546 /// <para></para>
547 /// </returns>
548 [MethodImpl(MethodImplOptions.AggressiveInlining)]
549 public int IndexOf(TLink item)
550 {
551     if (_equalityComparer.Equals(Index, item))
552     {
553         return _constants.IndexPart;
554     }
555     if (_equalityComparer.Equals(Source, item))
556     {
557         return _constants.SourcePart;
558     }
559     if (_equalityComparer.Equals(Target, item))
560     {
561         return _constants.TargetPart;
562     }
563     return -1;
564 }
565
566 /// <summary>
567 /// <para>
568 /// Inserts the index.
569 /// </para>
570 /// <para></para>
571 /// </summary>
572 /// <param name="index">
573 /// <para>The index.</para>
574 /// <para></para>
575 /// </param>
576 /// <param name="item">
577 /// <para>The item.</para>

```

```

578     /// <para></para>
579     /// </param>
580     [MethodImpl(MethodImplOptions.AggressiveInlining)]
581     public void Insert(int index, TLink item) => throw new NotSupportedException();
582
583     /// <summary>
584     /// <para>
585     /// Removes the at using the specified index.
586     /// </para>
587     /// <para></para>
588     /// </summary>
589     /// <param name="index">
590     /// <para>The index.</para>
591     /// <para></para>
592     /// </param>
593     [MethodImpl(MethodImplOptions.AggressiveInlining)]
594     public void RemoveAt(int index) => throw new NotSupportedException();
595
596     [MethodImpl(MethodImplOptions.AggressiveInlining)]
597     public static bool operator ==(Link<TLink> left, Link<TLink> right) =>
598         ↪ left.Equals(right);
599
600     [MethodImpl(MethodImplOptions.AggressiveInlining)]
601     public static bool operator !=(Link<TLink> left, Link<TLink> right) => !(left == right);
602     #endregion
603 }
604 }

```

1.23 ./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>

```

```

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

```

1.24 ./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 }

```

1.25 ./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 }

```

1.26 ./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.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.Memory
8  {
9      /// <summary>
10     /// <para>
11     /// Defines the links tree methods.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public interface ILinksTreeMethods<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Counts the usages using the specified root.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="root">
24         /// <para>The root.</para>
25         /// <para></para>
26         /// </param>
27         /// <returns>
28         /// <para>The link</para>
29         /// <para></para>
30         /// </returns>
31         [MethodImpl(MethodImplOptions.AggressiveInlining)]
32         TLink CountUsages(TLink root);
33
34         /// <summary>
35         /// <para>
36         /// Searches the source.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="source">
41         /// <para>The source.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="target">
45         /// <para>The target.</para>
46         /// <para></para>
47         /// </param>
48         /// <returns>
49         /// <para>The link</para>
50         /// <para></para>

```



```

51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     TLink Search(TLink source, TLink target);
54
55     /// <summary>
56     /// <para>
57     /// Eaches the usage using the specified root.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="root">
62     /// <para>The root.</para>
63     /// <para></para>
64     /// </param>
65     /// <param name="handler">
66     /// <para>The handler.</para>
67     /// <para></para>
68     /// </param>
69     /// <returns>
70     /// <para>The link</para>
71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     TLink EachUsage(TLink root, Func<IList<TLink>, TLink> handler);
75
76     /// <summary>
77     /// <para>
78     /// Detaches the root.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="root">
83     /// <para>The root.</para>
84     /// <para></para>
85     /// </param>
86     /// <param name="linkIndex">
87     /// <para>The link index.</para>
88     /// <para></para>
89     /// </param>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     void Detach(ref TLink root, TLink linkIndex);
92
93     /// <summary>
94     /// <para>
95     /// Attaches the root.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="root">
100    /// <para>The root.</para>
101    /// <para></para>
102    /// </param>
103    /// <param name="linkIndex">
104    /// <para>The link index.</para>
105    /// <para></para>
106    /// </param>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    void Attach(ref TLink root, TLink linkIndex);
109 }
110 }

```

1.27 ./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 }

```

1.28 ./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 }

```

```

51     /// The first free link.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     public TLink FirstFreeLink;
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,
139         ↪ FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();
140
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static bool operator ==(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
143         ↪ left.Equals(right);
144
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static bool operator !=(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
147         ↪ !(left == right);
148 }
149

```

1.29 `./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 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 external links recursionless size balanced tree methods base.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}"/>
20     /// <seealso cref="ILinksTreeMethods{TLink}"/>
21     public unsafe abstract class ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
22     ↪ RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
23     {
24         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
25         ↪ UncheckedConverter<TLink, long>.Default;
26
27         /// <summary>
28         /// <para>
29         /// The break.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected readonly TLink Break;
34         /// <summary>
35         /// <para>
36         /// The continue.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         protected readonly TLink Continue;
41         /// <summary>
42         /// <para>
43         /// The links data parts.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         protected readonly byte* LinksDataParts;
48         /// <summary>
49         /// <para>
50         /// The links index parts.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         protected readonly byte* LinksIndexParts;
55         /// <summary>
56         /// <para>
57         /// The links index parts.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         protected readonly byte* LinksIndexParts;
62     }
63 }

```

```

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

```

```

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

151
152    /// <summary>
153    /// <para>
154    /// Determines whether this instance first is to the left of second.
155    /// </para>
156    /// <para></para>
157    /// </summary>
158    /// <param name="source">
159    /// <para>The source.</para>
160    /// <para></para>
161    /// </param>
162    /// <param name="target">
163    /// <para>The target.</para>
164    /// <para></para>
165    /// </param>
166    /// <param name="rootSource">
167    /// <para>The root source.</para>
168    /// <para></para>
169    /// </param>
170    /// <param name="rootTarget">
171    /// <para>The root target.</para>
172    /// <para></para>
173    /// </param>
174    /// <returns>
175    /// <para>The bool</para>
176    /// <para></para>
177    /// </returns>
178    [MethodImpl(MethodImplOptions.AggressiveInlining)]
179    protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);

180
181    /// <summary>
182    /// <para>
183    /// Gets the header reference.
184    /// </para>
185    /// <para></para>
186    /// </summary>
187    /// <returns>
188    /// <para>A ref links header of t link</para>
189    /// <para></para>
190    /// </returns>
191    [MethodImpl(MethodImplOptions.AggressiveInlining)]
192    protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLink>>(Header);

193
194    /// <summary>
195    /// <para>
196    /// Gets the link data part reference using the specified link.
197    /// </para>
198    /// <para></para>
199    /// </summary>
200    /// <param name="link">
201    /// <para>The link.</para>
202    /// <para></para>
203    /// </param>
204    /// <returns>

```

```

205 /// <para>A ref raw link data part of t link</para>
206 /// <para></para>
207 /// </returns>
208 [MethodImpl(MethodImplOptions.AggressiveInlining)]
209 protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↳ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));

210
211 /// <summary>
212 /// <para>
213 /// Gets the link index part reference using the specified link.
214 /// </para>
215 /// <para></para>
216 /// </summary>
217 /// <param name="link">
218 /// <para>The link.</para>
219 /// <para></para>
220 /// </param>
221 /// <returns>
222 /// <para>A ref raw link index part of t link</para>
223 /// <para></para>
224 /// </returns>
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↳ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
    ↳ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));

227
228 /// <summary>
229 /// <para>
230 /// Gets the link values using the specified link index.
231 /// </para>
232 /// <para></para>
233 /// </summary>
234 /// <param name="linkIndex">
235 /// <para>The link index.</para>
236 /// <para></para>
237 /// </param>
238 /// <returns>
239 /// <para>A list of t link</para>
240 /// <para></para>
241 /// </returns>
242 [MethodImpl(MethodImplOptions.AggressiveInlining)]
243 protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
244 {
245     ref var link = ref GetLinkDataPartReference(linkIndex);
246     return new Link<TLink>(linkIndex, link.Source, link.Target);
247 }
248
249 /// <summary>
250 /// <para>
251 /// Determines whether this instance first is to the left of second.
252 /// </para>
253 /// <para></para>
254 /// </summary>
255 /// <param name="first">
256 /// <para>The first.</para>
257 /// <para></para>
258 /// </param>
259 /// <param name="second">
260 /// <para>The second.</para>
261 /// <para></para>
262 /// </param>
263 /// <returns>
264 /// <para>The bool</para>
265 /// <para></para>
266 /// </returns>
267 [MethodImpl(MethodImplOptions.AggressiveInlining)]
268 protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
269 {
270     ref var firstLink = ref GetLinkDataPartReference(first);
271     ref var secondLink = ref GetLinkDataPartReference(second);
272     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↳ secondLink.Source, secondLink.Target);
273 }
274
275 /// <summary>
276 /// <para>
277 /// Determines whether this instance first is to the right of second.

```

```

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

```



```

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

```

```

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

```



```

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

```

```

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

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

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

193     /// <summary>
194     /// <para>
195     /// Gets the link data part reference using the specified link.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// </summary>

```

```

200 /// <param name="link">
201 /// <para>The link.</para>
202 /// <para></para>
203 /// </param>
204 /// <returns>
205 /// <para>A ref raw link data part of t link</para>
206 /// <para></para>
207 /// </returns>
208 [MethodImpl(MethodImplOptions.AggressiveInlining)]
209 protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↳ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));

210
211 /// <summary>
212 /// <para>
213 /// Gets the link index part reference using the specified link.
214 /// </para>
215 /// <para></para>
216 /// </summary>
217 /// <param name="link">
218 /// <para>The link.</para>
219 /// <para></para>
220 /// </param>
221 /// <returns>
222 /// <para>A ref raw link index part of t link</para>
223 /// <para></para>
224 /// </returns>
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↳ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
    ↳ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));

227
228 /// <summary>
229 /// <para>
230 /// Gets the link values using the specified link index.
231 /// </para>
232 /// <para></para>
233 /// </summary>
234 /// <param name="linkIndex">
235 /// <para>The link index.</para>
236 /// <para></para>
237 /// </param>
238 /// <returns>
239 /// <para>A list of t link</para>
240 /// <para></para>
241 /// </returns>
242 [MethodImpl(MethodImplOptions.AggressiveInlining)]
243 protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
244 {
245     ref var link = ref GetLinkDataPartReference(linkIndex);
246     return new Link<TLink>(linkIndex, link.Source, link.Target);
247 }
248
249 /// <summary>
250 /// <para>
251 /// Determines whether this instance first is to the left of second.
252 /// </para>
253 /// <para></para>
254 /// </summary>
255 /// <param name="first">
256 /// <para>The first.</para>
257 /// <para></para>
258 /// </param>
259 /// <param name="second">
260 /// <para>The second.</para>
261 /// <para></para>
262 /// </param>
263 /// <returns>
264 /// <para>The bool</para>
265 /// <para></para>
266 /// </returns>
267 [MethodImpl(MethodImplOptions.AggressiveInlining)]
268 protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
269 {
270     ref var firstLink = ref GetLinkDataPartReference(first);
271     ref var secondLink = ref GetLinkDataPartReference(second);
272     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
    ↳ secondLink.Source, secondLink.Target);

```

```

273 }
274
275 /// <summary>
276 /// <para>
277 /// Determines whether this instance first is to the right of second.
278 /// </para>
279 /// <para></para>
280 /// </summary>
281 /// <param name="first">
282 /// <para>The first.</para>
283 /// <para></para>
284 /// </param>
285 /// <param name="second">
286 /// <para>The second.</para>
287 /// <para></para>
288 /// </param>
289 /// <returns>
290 /// <para>The bool</para>
291 /// <para></para>
292 /// </returns>
293 [MethodImpl(MethodImplOptions.AggressiveInlining)]
294 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
295 {
296     ref var firstLink = ref GetLinkDataPartReference(first);
297     ref var secondLink = ref GetLinkDataPartReference(second);
298     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
299         ↪ 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
335             ↪ broken)
336     }
337 }
338
339 /// <summary>
340 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
341 ↪ (концом).
342 /// </summary>
343 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
344 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
345 /// <returns>Индекс искомой связи.</returns>
346 [MethodImpl(MethodImplOptions.AggressiveInlining)]
347 public TLink Search(TLink source, TLink target)
348 {
349     var root = GetTreeRoot();
350     while (!EqualToZero(root))

```

```

348     {
349         ref var rootLink = ref GetLinkDataPartReference(root);
350         var rootSource = rootLink.Source;
351         var rootTarget = rootLink.Target;
352         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
353             ↳ node.Key < root.Key
354             {
355                 root = GetLeftOrDefault(root);
356             }
357         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
358             ↳ node.Key > root.Key
359             {
360                 root = GetRightOrDefault(root);
361             }
362         else // node.Key == root.Key
363         {
364             return root;
365         }
366     }
367     return Zero;
368 }
369
370 // TODO: Return indices range instead of references count
371 /// <summary>
372 /// <para>
373 /// Counts the usages using the specified link.
374 /// </para>
375 /// <para></para>
376 /// </summary>
377 /// <param name="link">
378 /// <para>The link.</para>
379 /// <para></para>
380 /// </param>
381 /// <returns>
382 /// <para>The link</para>
383 /// <para></para>
384 /// </returns>
385 [MethodImpl(MethodImplOptions.AggressiveInlining)]
386 public 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>

```



```

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

```

```

500         return @break;
501     }
502 }
503 return @continue;
504 }
505
506 /// <summary>
507 /// <para>
508 /// Prints the node value using the specified node.
509 /// </para>
510 /// <para></para>
511 /// </summary>
512 /// <param name="node">
513 /// <para>The node.</para>
514 /// <para></para>
515 /// </param>
516 /// <param name="sb">
517 /// <para>The sb.</para>
518 /// <para></para>
519 /// </param>
520 [MethodImpl(MethodImplOptions.AggressiveInlining)]
521 protected override void PrintNodeValue(TLink node, StringBuilder sb)
522 {
523     ref var link = ref GetLinkDataPartReference(node);
524     sb.Append(' ');
525     sb.Append(link.Source);
526     sb.Append('-');
527     sb.Append('>');
528     sb.Append(link.Target);
529 }
530 }
531 }

```

1.31 ./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> :
15        ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↪ cref="ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="linksDataParts">
29        /// <para>A links data parts.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="linksIndexParts">
33        /// <para>A links index parts.</para>
34        /// <para></para>
35        /// </param>
36        /// <param name="header">
37        /// <para>A header.</para>
38        /// <para></para>
39        /// </param>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]
41        public ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
42            ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
43            ↪ base(constants, linksDataParts, linksIndexParts, header) { }
44    }
45 }

```

```

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) =>
    ↪ GetLinkIndexPartReference(node).RightAsSource;
108
109    /// <summary>
110    /// <para>
111    /// Sets the left using the specified node.
112    /// </para>
113    /// <para></para>
114    /// </summary>

```

```

115     /// <param name="node">
116     /// <para>The node.</para>
117     /// <para></para>
118     /// </param>
119     /// <param name="left">
120     /// <para>The left.</para>
121     /// <para></para>
122     /// </param>
123     [MethodImpl(MethodImplOptions.AggressiveInlining)]
124     protected override void SetLeft(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.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <returns>
188     /// <para>The link</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]
192     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;

```

```

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

```

1.32 ./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
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>
121    /// <para></para>
122    /// </param>
123    /// <param name="left">
124    /// <para>The left.</para>
125    /// <para></para>
126    /// </param>
127    [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     /// </summary>
132     /// <param name="node">
133     /// <para>The node.</para>
134     /// </param>
135     /// <param name="right">
136     /// <para>The right.</para>
137     /// </param>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     protected override void SetRight(TLink node, TLink right) =>
140         ↪ GetLinkIndexPartReference(node).RightAsSource = right;
141
142     /// <summary>
143     /// <para>
144     /// Gets the size using the specified node.
145     /// </para>
146     /// </summary>
147     /// <param name="node">
148     /// <para>The node.</para>
149     /// </param>
150     /// <returns>
151     /// <para>The link</para>
152     /// </returns>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     protected override TLink GetSize(TLink node) =>
155         ↪ GetLinkIndexPartReference(node).SizeAsSource;
156
157     /// <summary>
158     /// <para>
159     /// Sets the size using the specified node.
160     /// </para>
161     /// </summary>
162     /// <param name="node">
163     /// <para>The node.</para>
164     /// </param>
165     /// <param name="size">
166     /// <para>The size.</para>
167     /// </param>
168     [MethodImpl(MethodImplOptions.AggressiveInlining)]
169     protected override void SetSize(TLink node, TLink size) =>
170         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
171
172     /// <summary>
173     /// <para>
174     /// Gets the tree root.
175     /// </para>
176     /// </summary>
177     /// <returns>
178     /// <para>The link</para>
179     /// </returns>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
182
183     /// <summary>
184     /// <para>
185     /// Gets the base part value using the specified link.
186     /// </para>
187     /// </summary>
188     /// <param name="link">
189     /// <para>The link.</para>
190

```



```

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

```

1.33 ./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> :
15    ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        ↪ Initializes a new <see
20        ↪ cref="ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="linksDataParts">
29        /// <para>A links data parts.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="linksIndexParts">
33        /// <para>A links index parts.</para>
34        /// <para></para>
35        /// </param>
36        /// <param name="header">
37        /// <para>A header.</para>
38        /// <para></para>
39        /// </param>
40        [MethodImpl(MethodImplOptions.AggressiveInlining)]
41        public ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
42        ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
43        ↪ base(constants, linksDataParts, linksIndexParts, header) { }
44
45        /// <summary>
46        /// <para>
47        /// Gets the left reference using the specified node.
48        /// </para>
49        /// <para></para>
50        /// </summary>
51        /// <param name="node">
52        /// <para>The node.</para>
53        /// <para></para>
54        /// </param>
55        /// <returns>
56        /// <para>The ref link</para>
57        /// <para></para>
58        /// </returns>
59        [MethodImpl(MethodImplOptions.AggressiveInlining)]
60        protected override ref TLink GetLeftReference(TLink node) => ref
61        ↪ 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
74     ↪ GetLinkIndexPartReference(node).RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) =>
92     ↪ GetLinkIndexPartReference(node).LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLink GetRight(TLink node) =>
110    ↪ GetLinkIndexPartReference(node).RightAsTarget;
111
112    /// <summary>
113    /// <para>
114    /// Sets the left using the specified node.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="node">
119    /// <para>The node.</para>
120    /// <para></para>
121    /// </param>
122    /// <param name="left">
123    /// <para>The left.</para>
124    /// <para></para>
125    /// </param>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLink node, TLink left) =>
128    ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <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>
185     /// </summary>
186     /// <returns>
187     /// <para>The link</para>
188     /// <para></para>
189     /// </returns>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
192
193     /// <summary>
194     /// <para>
195     /// Gets the base part value using the specified link.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="link">
200     /// <para>The link.</para>
201     /// <para></para>
202     /// </param>
203     /// <returns>
204     /// <para>The link</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

1.34 ./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> :
15         ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="ExternalLinksTargetsSizeBalancedTreeMethods" /> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// </param>
26         /// <param name="linksDataParts">
27         /// <para>A links data parts.</para>
28         /// </param>
29         /// <param name="linksIndexParts">
30         /// <para>A links index parts.</para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
37             ↪ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
38             ↪ linksDataParts, linksIndexParts, 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         /// </param>
49         /// <returns>
50         /// <para>The ref link</para>
51         /// </returns>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         protected override ref TLink GetLeftReference(TLink node) => ref
54             ↪ GetLinkIndexPartReference(node).LeftAsTarget;
55
56         /// <summary>
57         /// <para>
58         /// Gets the right reference using the specified node.
59         /// </para>
60         /// </summary>
61         /// <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
74         ↪ GetLinkIndexPartReference(node).RightAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the left using the specified node.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="node">
83     /// <para>The node.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>The link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override TLink GetLeft(TLink node) =>
92         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
93
94     /// <summary>
95     /// <para>
96     /// Gets the right using the specified node.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="node">
101    /// <para>The node.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected override TLink GetRight(TLink node) =>
110        ↪ GetLinkIndexPartReference(node).RightAsTarget;
111
112    /// <summary>
113    /// <para>
114    /// Sets the left using the specified node.
115    /// </para>
116    /// <para></para>
117    /// </summary>
118    /// <param name="node">
119    /// <para>The node.</para>
120    /// <para></para>
121    /// </param>
122    /// <param name="left">
123    /// <para>The left.</para>
124    /// <para></para>
125    /// </param>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLink node, TLink left) =>
128        ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
129
130    /// <summary>
131    /// <para>
132    /// Sets the right using the specified node.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">
141    /// <para>The right.</para>
142    /// <para></para>
143    /// </param>

```

```

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     /// </param>
153     /// <returns>
154     /// <para>The link</para>
155     /// <para></para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override TLink GetSize(TLink node) =>
159         ↪ GetLinkIndexPartReference(node).SizeAsTarget;
160
161     /// <summary>
162     /// <para>
163     /// Sets the size using the specified node.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="node">
168     /// <para>The node.</para>
169     /// </param>
170     /// <param name="size">
171     /// <para>The size.</para>
172     /// </param>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected override void SetSize(TLink node, TLink size) =>
175         ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
176
177     /// <summary>
178     /// <para>
179     /// Gets the tree root.
180     /// </para>
181     /// <para></para>
182     /// </summary>
183     /// <returns>
184     /// <para>The link</para>
185     /// <para></para>
186     /// </returns>
187     [MethodImpl(MethodImplOptions.AggressiveInlining)]
188     protected override 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     /// </param>
199     /// <returns>
200     /// <para>The link</para>
201     /// <para></para>
202     /// </returns>
203     [MethodImpl(MethodImplOptions.AggressiveInlining)]
204     protected override TLink GetBasePartValue(TLink link) =>
205         ↪ GetLinkDataPartReference(link).Target;
206
207     /// <summary>
208     /// <para>
209     /// Determines whether this instance first is to the left of second.
210     /// </para>
211     /// <para></para>
212     /// </summary>

```



```

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

```

1.35 ./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 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 recursionless size balanced tree methods base.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}" />
20     /// <seealso cref="ILinksTreeMethods{TLink}" />
21     public unsafe abstract class InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
22     ↪ RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
23     {
24         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
25         ↪ UncheckedConverter<TLink, long>.Default;
26
27         /// <summary>
28         /// <para>
29         /// The break.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected readonly TLink Break;
34
35         /// <summary>
36         /// <para>
37         /// The continue.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         protected readonly TLink Continue;
42
43         /// <summary>
44         /// <para>
45         /// The links data parts.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         protected readonly byte* LinksDataParts;
50
51         /// <summary>
52         /// <para>
53         /// The links index parts.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         protected readonly byte* LinksIndexParts;
58
59         /// <summary>
60         /// <para>
61         /// The header.
62         /// </para>
63         /// <para></para>
64         /// </summary>
65         protected readonly byte* Header;
66
67         /// <summary>
68         /// <para>
69         /// Initializes a new <see
70         ↪ cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase" /> 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">

```

```

76     /// <para>A links index parts.</para>
77     /// <para></para>
78     /// </param>
79     /// <param name="header">
80     /// <para>A header.</para>
81     /// <para></para>
82     /// </param>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     protected InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
85     ↪ 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     /// <summary>
94     /// <para>
95     /// Gets the tree root using the specified link.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="link">
100    /// <para>The link.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected abstract TLink GetTreeRoot(TLink link);
109
110    /// <summary>
111    /// <para>
112    /// Gets the base part value using the specified link.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="link">
117    /// <para>The link.</para>
118    /// <para></para>
119    /// </param>
120    /// <returns>
121    /// <para>The link</para>
122    /// <para></para>
123    /// </returns>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected abstract TLink GetBasePartValue(TLink link);
126
127    /// <summary>
128    /// <para>
129    /// Gets the key part value using the specified link.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="link">
134    /// <para>The link.</para>
135    /// <para></para>
136    /// </param>
137    /// <returns>
138    /// <para>The link</para>
139    /// <para></para>
140    /// </returns>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected abstract TLink GetKeyPartValue(TLink link);
143
144    /// <summary>
145    /// <para>
146    /// Gets the link data part reference using the specified link.
147    /// </para>
148    /// <para></para>
149    /// </summary>
150    /// <param name="link">
151    /// <para>The link.</para>
152    /// <para></para>

```

```

153     /// </param>
154     /// <returns>
155     /// <para>A ref raw link data part of t link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↳ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
        ↳ _addressToInt64Converter.Convert(link)));

160
161     /// <summary>
162     /// <para>
163     /// Gets the link index part reference using the specified link.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="link">
168     /// <para>The link.</para>
169     /// <para></para>
170     /// </param>
171     /// <returns>
172     /// <para>A ref raw link index part of t link</para>
173     /// <para></para>
174     /// </returns>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↳ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
        ↳ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));

177
178     /// <summary>
179     /// <para>
180     /// Determines whether this instance first is to the left of second.
181     /// </para>
182     /// <para></para>
183     /// </summary>
184     /// <param name="first">
185     /// <para>The first.</para>
186     /// <para></para>
187     /// </param>
188     /// <param name="second">
189     /// <para>The second.</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 FirstIsToLeftOfSecond(TLink first, TLink second) =>
        ↳ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));

198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance first is to the right of second.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="first">
206     /// <para>The first.</para>
207     /// <para></para>
208     /// </param>
209     /// <param name="second">
210     /// <para>The second.</para>
211     /// <para></para>
212     /// </param>
213     /// <returns>
214     /// <para>The bool</para>
215     /// <para></para>
216     /// </returns>
217     [MethodImpl(MethodImplOptions.AggressiveInlining)]
218     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
        ↳ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));

219
220     /// <summary>
221     /// <para>
222     /// Gets the link values using the specified link index.
223     /// </para>
224     /// <para></para>

```

```

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

```

```

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

```

```

377     /// <para>The link.</para>
378     /// <para></para>
379     /// </param>
380     /// <param name="handler">
381     /// <para>The handler.</para>
382     /// <para></para>
383     /// </param>
384     /// <returns>
385     /// <para>The continue.</para>
386     /// <para></para>
387     /// </returns>
388     [MethodImpl(MethodImplOptions.AggressiveInlining)]
389     private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
390     {
391         var @continue = Continue;
392         if (EqualToZero(link))
393         {
394             return @continue;
395         }
396         var @break = Break;
397         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
398         {
399             return @break;
400         }
401         if (AreEqual(handler(GetLinkValues(link)), @break))
402         {
403             return @break;
404         }
405         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
406         {
407             return @break;
408         }
409         return @continue;
410     }
411
412     /// <summary>
413     /// <para>
414     /// Prints the node value using the specified node.
415     /// </para>
416     /// <para></para>
417     /// </summary>
418     /// <param name="node">
419     /// <para>The node.</para>
420     /// <para></para>
421     /// </param>
422     /// <param name="sb">
423     /// <para>The sb.</para>
424     /// <para></para>
425     /// </param>
426     [MethodImpl(MethodImplOptions.AggressiveInlining)]
427     protected override void PrintNodeValue(TLink node, StringBuilder sb)
428     {
429         ref var link = ref GetLinkDataPartReference(node);
430         sb.Append(' ');
431         sb.Append(link.Source);
432         sb.Append('-');
433         sb.Append('>');
434         sb.Append(link.Target);
435     }
436 }
437 }

```

1.36 ./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 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 size balanced tree methods base.
16     /// </para>

```

```

17  /// <para></para>
18  /// </summary>
19  /// <seealso cref="SizeBalancedTreeMethods{TLink}"/>
20  /// <seealso cref="ILinksTreeMethods{TLink}"/>
21  public unsafe abstract class InternalLinksSizeBalancedTreeMethodsBase<TLink> :
    ↳ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
22  {
23      private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
    ↳ UncheckedConverter<TLink, long>.Default;
24
25      /// <summary>
26      /// <para>
27      /// The break.
28      /// </para>
29      /// <para></para>
30      /// </summary>
31      protected readonly TLink Break;
32      /// <summary>
33      /// <para>
34      /// The continue.
35      /// </para>
36      /// <para></para>
37      /// </summary>
38      protected readonly TLink Continue;
39      /// <summary>
40      /// <para>
41      /// The links data parts.
42      /// </para>
43      /// <para></para>
44      /// </summary>
45      protected readonly byte* LinksDataParts;
46      /// <summary>
47      /// <para>
48      /// The links index parts.
49      /// </para>
50      /// <para></para>
51      /// </summary>
52      protected readonly byte* LinksIndexParts;
53      /// <summary>
54      /// <para>
55      /// The header.
56      /// </para>
57      /// <para></para>
58      /// </summary>
59      protected readonly byte* Header;
60
61      /// <summary>
62      /// <para>
63      /// Initializes a new <see cref="InternalLinksSizeBalancedTreeMethodsBase"/> instance.
64      /// </para>
65      /// <para></para>
66      /// </summary>
67      /// <param name="constants">
68      /// <para>A constants.</para>
69      /// <para></para>
70      /// </param>
71      /// <param name="linksDataParts">
72      /// <para>A links data parts.</para>
73      /// <para></para>
74      /// </param>
75      /// <param name="linksIndexParts">
76      /// <para>A links index parts.</para>
77      /// <para></para>
78      /// </param>
79      /// <param name="header">
80      /// <para>A header.</para>
81      /// <para></para>
82      /// </param>
83      [MethodImpl(MethodImplOptions.AggressiveInlining)]
84      protected InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
    ↳ byte* linksDataParts, byte* linksIndexParts, byte* header)
85      {
86          LinksDataParts = linksDataParts;
87          LinksIndexParts = linksIndexParts;
88          Header = header;
89          Break = constants.Break;
90          Continue = constants.Continue;
91      }
92

```



```

93     /// <summary>
94     /// <para>
95     /// Gets the tree root using the specified link.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="link">
100    /// <para>The link.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected abstract TLink GetTreeRoot(TLink link);
109
110    /// <summary>
111    /// <para>
112    /// Gets the base part value using the specified link.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <param name="link">
117    /// <para>The link.</para>
118    /// <para></para>
119    /// </param>
120    /// <returns>
121    /// <para>The link</para>
122    /// <para></para>
123    /// </returns>
124    [MethodImpl(MethodImplOptions.AggressiveInlining)]
125    protected abstract TLink GetBasePartValue(TLink link);
126
127    /// <summary>
128    /// <para>
129    /// Gets the key part value using the specified link.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="link">
134    /// <para>The link.</para>
135    /// <para></para>
136    /// </param>
137    /// <returns>
138    /// <para>The link</para>
139    /// <para></para>
140    /// </returns>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected abstract TLink GetKeyPartValue(TLink link);
143
144    /// <summary>
145    /// <para>
146    /// Gets the link data part reference using the specified link.
147    /// </para>
148    /// <para></para>
149    /// </summary>
150    /// <param name="link">
151    /// <para>The link.</para>
152    /// <para></para>
153    /// </param>
154    /// <returns>
155    /// <para>A ref raw link data part of t link</para>
156    /// <para></para>
157    /// </returns>
158    [MethodImpl(MethodImplOptions.AggressiveInlining)]
159    protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↪ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
        ↪ _addressToInt64Converter.Convert(link)));
160
161    /// <summary>
162    /// <para>
163    /// Gets the link index part reference using the specified link.
164    /// </para>
165    /// <para></para>
166    /// </summary>
167    /// <param name="link">
168    /// <para>The link.</para>

```

```

169    /// <para></para>
170    /// </param>
171    /// <returns>
172    /// <para>A ref raw link index part of t link</para>
173    /// <para></para>
174    /// </returns>
175    [MethodImpl(MethodImplOptions.AggressiveInlining)]
176    protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
177        ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
178        ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
179
180    /// <summary>
181    /// <para>
182    /// <para>Determines whether this instance first is to the left of second.
183    /// </para>
184    /// <para></para>
185    /// </summary>
186    /// <param name="first">
187    /// <para>The first.</para>
188    /// <para></para>
189    /// </param>
190    /// <param name="second">
191    /// <para>The second.</para>
192    /// <para></para>
193    /// </param>
194    /// <returns>
195    /// <para>The bool</para>
196    /// <para></para>
197    /// </returns>
198    [MethodImpl(MethodImplOptions.AggressiveInlining)]
199    protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
200        ↪ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
201
202    /// <summary>
203    /// <para>
204    /// <para>Determines whether this instance first is to the right of second.
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 FirstIsToTheRightOfSecond(TLink first, TLink second) =>
222        ↪ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
223
224    /// <summary>
225    /// <para>
226    /// <para>Gets the link values using the specified link index.
227    /// </para>
228    /// <para></para>
229    /// </summary>
230    /// <param name="linkIndex">
231    /// <para>The link index.</para>
232    /// <para></para>
233    /// </param>
234    /// <returns>
235    /// <para>A list of t link</para>
236    /// <para></para>
237    /// </returns>
238    [MethodImpl(MethodImplOptions.AggressiveInlining)]
239    protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
240    {
241        ref var link = ref GetLinkDataPartReference(linkIndex);
242        return new Link<TLink>(linkIndex, link.Source, link.Target);
243    }
244
245    /// <summary>
246    /// <para>

```

```

243     /// The zero.
244     /// </para>
245     /// <para></para>
246     /// </summary>
247     public TLink this[TLink link, TLink index]
248     {
249         [MethodImpl(MethodImplOptions.AggressiveInlining)]
250         get
251         {
252             var root = GetTreeRoot(link);
253             if (GreaterOrEqualThan(index, GetSize(root)))
254             {
255                 return Zero;
256             }
257             while (!EqualToZero(root))
258             {
259                 var left = GetLeftOrDefault(root);
260                 var leftSize = GetSizeOrZero(left);
261                 if (LessThan(index, leftSize))
262                 {
263                     root = left;
264                     continue;
265                 }
266                 if (AreEqual(index, leftSize))
267                 {
268                     return root;
269                 }
270                 root = GetRightOrDefault(root);
271                 index = Subtract(index, Increment(leftSize));
272             }
273             return Zero; // TODO: Impossible situation exception (only if tree structure
                ↳ broken)
274         }
275     }
276
277     /// <summary>
278     /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
279     ↳ (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

```

```

319         {
320             return root;
321         }
322     }
323     return Zero;
324 }
325
326 // TODO: Return indices range instead of references count
327 /// <summary>
328 /// <para>
329 /// Counts the usages using the specified link.
330 /// </para>
331 /// <para></para>
332 /// </summary>
333 /// <param name="link">
334 /// <para>The link.</para>
335 /// <para></para>
336 /// </param>
337 /// <returns>
338 /// <para>The link</para>
339 /// <para></para>
340 /// </returns>
341 [MethodImpl(MethodImplOptions.AggressiveInlining)]
342 public TLink CountUsages(TLink link) => GetSizeOrZero(GetTreeRoot(link));
343
344 /// <summary>
345 /// <para>
346 /// Eaches the usage using the specified base.
347 /// </para>
348 /// <para></para>
349 /// </summary>
350 /// <param name="@base">
351 /// <para>The base.</para>
352 /// <para></para>
353 /// </param>
354 /// <param name="handler">
355 /// <para>The handler.</para>
356 /// <para></para>
357 /// </param>
358 /// <returns>
359 /// <para>The link</para>
360 /// <para></para>
361 /// </returns>
362 [MethodImpl(MethodImplOptions.AggressiveInlining)]
363 public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
364     ↳ EachUsageCore(@base, GetTreeRoot(@base), handler);
365
366 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
367     ↳ low-level MSIL stack.
368 /// <summary>
369 /// <para>
370 /// Eaches the usage core using the specified base.
371 /// </para>
372 /// <para></para>
373 /// </summary>
374 /// <param name="@base">
375 /// <para>The base.</para>
376 /// <para></para>
377 /// </param>
378 /// <param name="link">
379 /// <para>The link.</para>
380 /// <para></para>
381 /// </param>
382 /// <param name="handler">
383 /// <para>The handler.</para>
384 /// <para></para>
385 /// </param>
386 /// <returns>
387 /// <para>The continue.</para>
388 /// <para></para>
389 /// </returns>
390 [MethodImpl(MethodImplOptions.AggressiveInlining)]
391 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
392 {
393     var @continue = Continue;
394     if (EqualToZero(link))
395     {
396         return @continue;
397     }
398 }

```

```

395     }
396     var @break = Break;
397     if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
398     {
399         return @break;
400     }
401     if (AreEqual(handler(GetLinkValues(link)), @break))
402     {
403         return @break;
404     }
405     if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
406     {
407         return @break;
408     }
409     return @continue;
410 }
411
412 /// <summary>
413 /// <para>
414 /// Prints the node value using the specified node.
415 /// </para>
416 /// <para></para>
417 /// </summary>
418 /// <param name="node">
419 /// <para>The node.</para>
420 /// <para></para>
421 /// </param>
422 /// <param name="sb">
423 /// <para>The sb.</para>
424 /// <para></para>
425 /// </param>
426 [MethodImpl(MethodImplOptions.AggressiveInlining)]
427 protected override void PrintNodeValue(TLink node, StringBuilder sb)
428 {
429     ref var link = ref GetLinkDataPartReference(node);
430     sb.Append(' ');
431     sb.Append(link.Source);
432     sb.Append('-');
433     sb.Append('>');
434     sb.Append(link.Target);
435 }
436 }
437 }

```

1.37 ./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 static System.Runtime.CompilerServices.Unsafe;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Generic
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the internal links sources linked list methods.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="RelativeCircularDoublyLinkedListMethods{TLink}"/>
19     public unsafe class InternalLinksSourcesLinkedListMethods<TLink> :
20         RelativeCircularDoublyLinkedListMethods<TLink>
21     {
22         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
23             UnsafeConverter<TLink, long>.Default;
24         private readonly byte* _linksDataParts;
25         private readonly byte* _linksIndexParts;
26
27         /// <summary>
28         /// <para>
29         /// The break.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected readonly TLink Break;
34
35         /// <summary>
36         /// <para>

```

```

33     /// The continue.
34     /// </para>
35     /// <para></para>
36     /// </summary>
37     protected readonly TLink Continue;
38
39     /// <summary>
40     /// <para>
41     /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
42     /// </para>
43     /// <para></para>
44     /// </summary>
45     /// <param name="constants">
46     /// <para>A constants.</para>
47     /// <para></para>
48     /// </param>
49     /// <param name="linksDataParts">
50     /// <para>A links data parts.</para>
51     /// <para></para>
52     /// </param>
53     /// <param name="linksIndexParts">
54     /// <para>A links index parts.</para>
55     /// <para></para>
56     /// </param>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants, byte*
59     ↪ 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
83     ↪ AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (RawLinkDataPart<TLink>.SizeInBytes
84     ↪ * _addressToInt64Converter.Convert(link)));
85
86     /// <summary>
87     /// <para>
88     /// Gets the link index part reference using the specified link.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="link">
93     /// <para>The link.</para>
94     /// <para></para>
95     /// </param>
96     /// <returns>
97     /// <para>A ref raw link index part of t link</para>
98     /// <para></para>
99     /// </returns>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
102    ↪ ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
103    ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
104
105    /// <summary>
106    /// <para>
107    /// Gets the first using the specified head.
108    /// </para>
109    /// <para></para>
110    /// </summary>

```

```

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

```

```

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

```



```

256 protected override void SetPrevious(TLink element, TLink previous) =>
    ↪ GetLinkIndexPartReference(element).LeftAsSource = previous;
257
258 /// <summary>
259 /// <para>
260 /// Sets the next using the specified element.
261 /// </para>
262 /// <para></para>
263 /// </summary>
264 /// <param name="element">
265 /// <para>The element.</para>
266 /// <para></para>
267 /// </param>
268 /// <param name="next">
269 /// <para>The next.</para>
270 /// <para></para>
271 /// </param>
272 [MethodImpl(MethodImplOptions.AggressiveInlining)]
273 protected override void SetNext(TLink element, TLink next) =>
    ↪ GetLinkIndexPartReference(element).RightAsSource = next;
274
275 /// <summary>
276 /// <para>
277 /// Sets the size using the specified head.
278 /// </para>
279 /// <para></para>
280 /// </summary>
281 /// <param name="head">
282 /// <para>The head.</para>
283 /// <para></para>
284 /// </param>
285 /// <param name="size">
286 /// <para>The size.</para>
287 /// <para></para>
288 /// </param>
289 [MethodImpl(MethodImplOptions.AggressiveInlining)]
290 protected override void SetSize(TLink head, TLink size) =>
    ↪ GetLinkIndexPartReference(head).SizeAsSource = size;
291
292 /// <summary>
293 /// <para>
294 /// Counts the usages using the specified head.
295 /// </para>
296 /// <para></para>
297 /// </summary>
298 /// <param name="head">
299 /// <para>The head.</para>
300 /// <para></para>
301 /// </param>
302 /// <returns>
303 /// <para>The link</para>
304 /// <para></para>
305 /// </returns>
306 [MethodImpl(MethodImplOptions.AggressiveInlining)]
307 public TLink CountUsages(TLink head) => GetSize(head);
308
309 /// <summary>
310 /// <para>
311 /// Gets the link values using the specified link index.
312 /// </para>
313 /// <para></para>
314 /// </summary>
315 /// <param name="linkIndex">
316 /// <para>The link index.</para>
317 /// <para></para>
318 /// </param>
319 /// <returns>
320 /// <para>A list of t link</para>
321 /// <para></para>
322 /// </returns>
323 [MethodImpl(MethodImplOptions.AggressiveInlining)]
324 protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
325 {
326     ref var link = ref GetLinkDataPartReference(linkIndex);
327     return new Link<TLink>(linkIndex, link.Source, link.Target);
328 }
329
330 /// <summary>

```

```

331     /// <para>
332     /// Eaches the usage using the specified source.
333     /// </para>
334     /// <para></para>
335     /// </summary>
336     /// <param name="source">
337     /// <para>The source.</para>
338     /// <para></para>
339     /// </param>
340     /// <param name="handler">
341     /// <para>The handler.</para>
342     /// <para></para>
343     /// </param>
344     /// <returns>
345     /// <para>The continue.</para>
346     /// <para></para>
347     /// </returns>
348     [MethodImpl(MethodImplOptions.AggressiveInlining)]
349     public TLink EachUsage(TLink source, Func<IList<TLink>, TLink> handler)
350     {
351         var @continue = Continue;
352         var @break = Break;
353         var current = GetFirst(source);
354         var first = current;
355         while (!EqualToZero(current))
356         {
357             if (AreEqual(handler(GetLinkValues(current)), @break))
358             {
359                 return @break;
360             }
361             current = GetNext(current);
362             if (AreEqual(current, first))
363             {
364                 return @continue;
365             }
366         }
367         return @continue;
368     }
369 }
370 }

```

1.38 ./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">

```

```

35    /// <para>A header.</para>
36    /// <para></para>
37    /// </param>
38    [MethodImpl(MethodImplOptions.AggressiveInlining)]
39    public InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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>
170     /// <param name="node">
171     /// <para>The node.</para>
172     /// <para></para>
173     /// </param>
174     /// <param name="size">
175     /// <para>The size.</para>
176     /// <para></para>
177     /// </param>
178     [MethodImpl(MethodImplOptions.AggressiveInlining)]
179     protected override void SetSize(TLink node, TLink size) =>
180         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
181
182     /// <summary>
183     /// <para>

```

```

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

```

```

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(source), target);
266     }
267 }

```

1.39 ./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> :
        ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="InternalLinksSourcesSizeBalancedTreeMethods"/> instance.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <param name="constants">
23        /// <para>A constants.</para>
24        /// <para></para>
25        /// </param>
26        /// <param name="linksDataParts">
27        /// <para>A links data parts.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="linksIndexParts">
31        /// <para>A links index parts.</para>
32        /// <para></para>
33        /// </param>
34        /// <param name="header">
35        /// <para>A header.</para>
36        /// <para></para>
37        /// </param>
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        public InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<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
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>

```

```

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

```



```

206     /// <para></para>
207     /// </returns>
208     [MethodImpl(MethodImplOptions.AggressiveInlining)]
209     protected override TLink GetBasePartValue(TLink link) =>
210         ↪ GetLinkDataPartReference(link).Source;
211
212     /// <summary>
213     /// <para>
214     /// Gets the key part value using the specified link.
215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="link">
219     /// <para>The link.</para>
220     /// </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     /// </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     /// </param>
256     /// <param name="target">
257     /// <para>The target.</para>
258     /// </param>
259     /// <returns>
260     /// <para>The link</para>
261     /// <para></para>
262     /// </returns>
263     public override TLink Search(TLink source, TLink target) =>
264         ↪ SearchCore(GetTreeRoot(source), target);
265
266     }
267 }

```

1.40 ./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> :
    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
15  {
16      /// <summary>
17      /// <para>
18      /// Initializes a new <see
    ↪ cref="InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 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) =>
91         ↪ GetLinkIndexPartReference(node).LeftAsTarget;
92
93     /// <summary>
94     /// <para>
95     /// Gets the right using the specified node.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetRight(TLink node) =>
109        ↪ GetLinkIndexPartReference(node).RightAsTarget;
110
111    /// <summary>
112    /// <para>
113    /// Sets the left using the specified node.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="node">
118    /// <para>The node.</para>
119    /// <para></para>
120    /// </param>
121    /// <param name="left">
122    /// <para>The left.</para>
123    /// <para></para>
124    /// </param>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    protected override void SetLeft(TLink node, TLink left) =>
127        ↪ GetLinkIndexPartReference(node).LeftAsTarget = left;
128
129    /// <summary>
130    /// <para>
131    /// Sets the right using the specified node.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="node">
136    /// <para>The node.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="right">
140    /// <para>The right.</para>
141    /// <para></para>
142    /// </param>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override void SetRight(TLink node, TLink right) =>
145        ↪ GetLinkIndexPartReference(node).RightAsTarget = right;
146
147    /// <summary>
148    /// <para>
149    /// Gets the size using the specified node.
150    /// </para>
151    /// <para></para>
152    /// </summary>
153    /// <param name="node">
154    /// <para>The node.</para>
155    /// <para></para>
156    /// </param>
157    /// <returns>
158    /// <para>The link</para>
159    /// <para></para>
160    /// </returns>

```

```

157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 protected override TLink GetSize(TLink node) =>
    ↳ GetLinkIndexPartReference(node).SizeAsTarget;

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

176
177 /// <summary>
178 /// <para>
179 /// Gets the tree root using the specified link.
180 /// </para>
181 /// <para></para>
182 /// </summary>
183 /// <param name="link">
184 /// <para>The link.</para>
185 /// <para></para>
186 /// </param>
187 /// <returns>
188 /// <para>The link</para>
189 /// <para></para>
190 /// </returns>
191 [MethodImpl(MethodImplOptions.AggressiveInlining)]
192 protected override TLink GetTreeRoot(TLink link) =>
    ↳ 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) =>
    ↳ GetLinkDataPartReference(link).Target;

210
211 /// <summary>
212 /// <para>
213 /// Gets the key part value using the specified link.
214 /// </para>
215 /// <para></para>
216 /// </summary>
217 /// <param name="link">
218 /// <para>The link.</para>
219 /// <para></para>
220 /// </param>
221 /// <returns>
222 /// <para>The link</para>
223 /// <para></para>
224 /// </returns>
225 [MethodImpl(MethodImplOptions.AggressiveInlining)]
226 protected override TLink GetKeyPartValue(TLink link) =>
    ↳ GetLinkDataPartReference(link).Source;

227
228 /// <summary>

```

```

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

```

1.41 ./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> :
15        ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="InternalLinksTargetsSizeBalancedTreeMethods" /> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="linksDataParts">
28        /// <para>A links data parts.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="linksIndexParts">
32        /// <para>A links index parts.</para>
33        /// <para></para>
34        /// </param>
35        /// <param name="header">
36        /// <para>A header.</para>

```

```

36  /// <para></para>
37  /// </param>
38  [MethodImpl(MethodImplOptions.AggressiveInlining)]
39  public 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
    ↳ 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 using the specified link.
183     /// </para>
184     /// <para></para>

```

```

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

```



```

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 }

```

1.42 ./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>
19     {
20         private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
21         private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
24         private byte* _header;
25         private byte* _linksDataParts;
26         private byte* _linksIndexParts;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="SplitMemoryLinks" /> instance.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="dataMemory">
35         /// <para>A data memory.</para>
36         /// <para></para>
37         /// </param>
38         /// <param name="indexMemory">
39         /// <para>A index memory.</para>
40         /// <para></para>
41         /// </param>
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public SplitMemoryLinks(string dataMemory, string indexMemory) : this(new
            ↪ FileMappedResizableDirectMemory(dataMemory), new
            ↪ FileMappedResizableDirectMemory(indexMemory)) { }
44
45         /// <summary>
46         /// <para>
47         /// Initializes a new <see cref="SplitMemoryLinks" /> instance.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="dataMemory">
52         /// <para>A data memory.</para>
53         /// <para></para>
54         /// </param>
55         /// <param name="indexMemory">
56         /// <para>A index memory.</para>
57         /// <para></para>
58         /// </param>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
61
62         /// <summary>

```

```

63     /// <para>
64     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     /// <param name="dataMemory">
69     /// <para>A data memory.</para>
70     /// <para></para>
71     /// </param>
72     /// <param name="indexMemory">
73     /// <para>A index memory.</para>
74     /// <para></para>
75     /// </param>
76     /// <param name="memoryReservationStep">
77     /// <para>A memory reservation step.</para>
78     /// <para></para>
79     /// </param>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↪ memoryReservationStep, Default<LinksConstants<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></param>
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
            ↪ InternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
144          _createExternalSourceTreeMethods = () => new
            ↪ ExternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
145          _createInternalTargetTreeMethods = () => new
            ↪ InternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
146          _createExternalTargetTreeMethods = () => new
            ↪ ExternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
147      }
148      else
149      {
150          _createInternalSourceTreeMethods = () => new
            ↪ InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
151          _createExternalSourceTreeMethods = () => new
            ↪ ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
152          _createInternalTargetTreeMethods = () => new
            ↪ InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
153          _createExternalTargetTreeMethods = () => new
            ↪ ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
            ↪ _linksDataParts, _linksIndexParts, _header);
154      }
155      Init(dataMemory, indexMemory);
156  }
157
158  /// <summary>
159  /// <para>
160  /// Sets the pointers using the specified data memory.
161  /// </para>
162  /// </para></summary>
163  /// </summary>
164  /// <param name="dataMemory">
165  /// <para>The data memory.</para>
166  /// </para></param>
167  /// </param>
168  /// <param name="indexMemory">
169  /// <para>The index memory.</para>
170  /// </para></param>
171  /// </param>
172  [MethodImpl(MethodImplOptions.AggressiveInlining)]
173  protected override void SetPointers(IResizableDirectMemory dataMemory,
    ↪ IResizableDirectMemory indexMemory)
174  {
175      _linksDataParts = (byte*)dataMemory.Pointer;
176      _linksIndexParts = (byte*)indexMemory.Pointer;
177      _header = _linksIndexParts;
178      if (_useLinkedList)
179      {
180          InternalSourcesListMethods = new
            ↪ InternalLinksSourcesLinkedListMethods<TLink>(Constants, _linksDataParts,
            ↪ _linksIndexParts);
181      }
182      else
183      {
184          InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
185      }
186      ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
187      InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
188      ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();

```

```

189     UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_linksDataParts, _header);
190 }
191
192 /// <summary>
193 /// <para>
194 /// Resets the pointers.
195 /// </para>
196 /// <para></para>
197 /// </summary>
198 [MethodImpl(MethodImplOptions.AggressiveInlining)]
199 protected override void ResetPointers()
200 {
201     base.ResetPointers();
202     _linksDataParts = null;
203     _linksIndexParts = null;
204     _header = null;
205 }
206
207 /// <summary>
208 /// <para>
209 /// Gets the header reference.
210 /// </para>
211 /// <para></para>
212 /// </summary>
213 /// <returns>
214 /// <para>A ref links header of t link</para>
215 /// <para></para>
216 /// </returns>
217 [MethodImpl(MethodImplOptions.AggressiveInlining)]
218 protected override ref LinksHeader<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 }
259 }

```

1.43 ./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
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.Memory.Split.Generic
14 {
15     /// <summary>
16     /// <para>
17     /// Represents the split memory links base.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     /// <seealso cref="DisposableBase"/>
22     /// <seealso cref="ILinks{TLink}"/>
23     public abstract class SplitMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink>
24     {
25         private static readonly EqualityComparer<TLink> _equalityComparer =
26             EqualityComparer<TLink>.Default;
27         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
28         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
29             UncheckedConverter<TLink, long>.Default;
30         private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
31             UncheckedConverter<long, TLink>.Default;
32
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 LinkDataPartSizeInBytes = RawLinkDataPart<TLink>.SizeInBytes;
42
43         /// <summary>
44         /// <para>
45         /// The size in bytes.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         public static readonly long LinkIndexPartSizeInBytes =
50             RawLinkIndexPart<TLink>.SizeInBytes;
51
52         /// <summary>
53         /// <para>
54         /// The size in bytes.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
59
60         /// <summary>
61         /// <para>
62         /// The default links size step.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
67
68         /// <summary>
69         /// <para>
70         /// The data memory.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         protected readonly IResizableDirectMemory _dataMemory;
75
76         /// <summary>
77         /// <para>
78         /// The index memory.
79         /// </para>
80         /// <para></para>
81         /// </summary>
82         protected readonly IResizableDirectMemory _indexMemory;
83
84         /// <summary>
85         /// <para>
86         /// The use linked list.
87         /// </para>
88         /// </summary>
89         protected readonly IUseLinkedList _useLinkedList;
90     }
91 }

```

```

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            {
155                ref var header = ref GetHeaderReference();
156                return Subtract(header.AllocatedLinks, header.FreeLinks);
157            }
158        }
159    }
160
161    /// <summary>
162    /// <para>

```

```

159     /// Gets the constants value.
160     /// </para>
161     /// <para></para>
162     /// </summary>
163     public virtual LinksConstants<TLink> Constants
164     {
165         [MethodImpl(MethodImplOptions.AggressiveInlining)]
166         get;
167     }
168
169     /// <summary>
170     /// <para>
171     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
172     /// </para>
173     /// <para></para>
174     /// </summary>
175     /// <param name="dataMemory">
176     /// <para>A data memory.</para>
177     /// <para></para>
178     /// </param>
179     /// <param name="indexMemory">
180     /// <para>A index memory.</para>
181     /// <para></para>
182     /// </param>
183     /// <param name="memoryReservationStep">
184     /// <para>A memory reservation step.</para>
185     /// <para></para>
186     /// </param>
187     /// <param name="constants">
188     /// <para>A constants.</para>
189     /// <para></para>
190     /// </param>
191     /// <param name="useLinkedList">
192     /// <para>A use linked list.</para>
193     /// <para></para>
194     /// </param>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants, bool
        ↪ useLinkedList)
197     {
198         _dataMemory = dataMemory;
199         _indexMemory = indexMemory;
200         _dataMemoryReservationStepInBytes = memoryReservationStep * LinkDataPartSizeInBytes;
201         _indexMemoryReservationStepInBytes = memoryReservationStep *
        ↪ LinkIndexPartSizeInBytes;
202         _useLinkedList = useLinkedList;
203         Constants = constants;
204     }
205
206     /// <summary>
207     /// <para>
208     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
209     /// </para>
210     /// <para></para>
211     /// </summary>
212     /// <param name="dataMemory">
213     /// <para>A data memory.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="indexMemory">
217     /// <para>A index memory.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="memoryReservationStep">
221     /// <para>A memory reservation step.</para>
222     /// <para></para>
223     /// </param>
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
        ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance, useLinkedList: true)
        ↪ { }
226
227     /// <summary>
228     /// <para>
229     /// Inits the data memory.
230     /// </para>

```

```

231 /// <para></para>
232 /// </summary>
233 /// <param name="dataMemory">
234 /// <para>The data memory.</para>
235 /// <para></para>
236 /// </param>
237 /// <param name="indexMemory">
238 /// <para>The index memory.</para>
239 /// <para></para>
240 /// </param>
241 [MethodImpl(MethodImplOptions.AggressiveInlining)]
242 protected virtual void Init(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory)
243 {
244     // Read allocated links from header
245     if (indexMemory.ReservedCapacity < LinkHeaderSizeInBytes)
246     {
247         indexMemory.ReservedCapacity = LinkHeaderSizeInBytes;
248     }
249     SetPointers(dataMemory, indexMemory);
250     ref var header = ref GetHeaderReference();
251     var allocatedLinks = ConvertToInt64(header.AllocatedLinks);
252     // Adjust reserved capacity
253     var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
254     if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
255     {
256         minimumDataReservedCapacity = dataMemory.UsedCapacity;
257     }
258     if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
259     {
260         minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
261     }
262     var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
263     if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
264     {
265         minimumIndexReservedCapacity = indexMemory.UsedCapacity;
266     }
267     if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
268     {
269         minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
270     }
271     // Check for alignment
272     if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
273     {
274         minimumDataReservedCapacity = ((minimumDataReservedCapacity /
    ↪ _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 restrictions.
302    /// </para>
303    /// <para></para>
304    /// </summary>
305    /// <param name="restrictions">
306    /// <para>The restrictions.</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> restrictions)
319    {
320        // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
321        if (restrictions.Count == 0)
322        {
323            return Total;
324        }
325        var constants = Constants;
326        var any = constants.Any;
327        var index = restrictions[constants.IndexPart];
328        if (restrictions.Count == 1)
329        {
330            if (AreEqual(index, any))
331            {
332                return Total;
333            }
334            return Exists(index) ? GetOne() : GetZero();
335        }
336        if (restrictions.Count == 2)
337        {
338            var value = restrictions[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);

```

```

373         if (AreEqual(storedLinkValue.Source, value) ||
374             ⇨ AreEqual(storedLinkValue.Target, value))
375         {
376             return GetOne();
377         }
378         return GetZero();
379     }
380     if (restrictions.Count == 3)
381     {
382         var externalReferencesRange = constants.ExternalReferencesRange;
383         var source = restrictions[constants.SourcePart];
384         var target = restrictions[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                 }
449                 else
450                 {

```

```

447         if (_useLinkedList ||
448             ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
449             ↪ InternalTargetsTreeMethods.CountUsages(target)))
450         {
451             link = InternalTargetsTreeMethods.Search(source, target);
452         }
453         else
454         {
455             link = InternalSourcesTreeMethods.Search(source, target);
456         }
457     }
458     else
459     {
460         if (_useLinkedList ||
461             ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
462             ↪ InternalTargetsTreeMethods.CountUsages(target)))
463         {
464             link = InternalTargetsTreeMethods.Search(source, target);
465         }
466         else
467         {
468             link = InternalSourcesTreeMethods.Search(source, target);
469         }
470     }
471     return AreEqual(link, constants.Null) ? GetZero() : GetOne();
472 }
473 else
474 {
475     if (!Exists(index))
476     {
477         return GetZero();
478     }
479     if (AreEqual(source, any) && AreEqual(target, any))
480     {
481         return GetOne();
482     }
483     ref var storedLinkValue = ref GetLinkDataPartReference(index);
484     if (!AreEqual(source, any) && !AreEqual(target, any))
485     {
486         if (AreEqual(storedLinkValue.Source, source) &&
487             ↪ AreEqual(storedLinkValue.Target, target))
488         {
489             return GetOne();
490         }
491         return GetZero();
492     }
493     var value = default(TLink);
494     if (AreEqual(source, any))
495     {
496         value = target;
497     }
498     if (AreEqual(target, any))
499     {
500         value = source;
501     }
502     if (AreEqual(storedLinkValue.Source, value) ||
503         ↪ AreEqual(storedLinkValue.Target, value))
504     {
505         return GetOne();
506     }
507     return GetZero();
508 }
509 }
510 throw new NotSupportedException("Другие размеры и способы ограничений не
511     ↪ поддерживаются.");
512 }
513
514 /// <summary>
515 /// <para>
516 /// Eaches the handler.
517 /// </para>
518 /// <para></para>
519 /// </summary>
520 /// <param name="handler">
521 /// <para>The handler.</para>
522 /// <para></para>

```

```

518 /// </param>
519 /// <param name="restrictions">
520 /// <para>The restrictions.</para>
521 /// <para></para>
522 /// </param>
523 /// <exception cref="NotSupportedException">
524 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
525 /// <para></para>
526 /// </exception>
527 /// <returns>
528 /// <para>The link</para>
529 /// <para></para>
530 /// </returns>
531 [MethodImpl(MethodImplOptions.AggressiveInlining)]
532 public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
533 {
534     var constants = Constants;
535     var @break = constants.Break;
536     if (restrictions.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             return @break;
546         }
547         var @continue = constants.Continue;
548         var any = constants.Any;
549         var index = restrictions[constants.IndexPart];
550         if (restrictions.Count == 1)
551         {
552             if (AreEqual(index, any))
553             {
554                 return Each(handler, Array.Empty<TLink>());
555             }
556             if (!Exists(index))
557             {
558                 return @continue;
559             }
560             return handler(GetLinkStruct(index));
561         }
562         if (restrictions.Count == 2)
563         {
564             var value = restrictions[1];
565             if (AreEqual(index, any))
566             {
567                 if (AreEqual(value, any))
568                 {
569                     return Each(handler, Array.Empty<TLink>());
570                 }
571                 if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
572                 {
573                     return @break;
574                 }
575                 return Each(handler, new Link<TLink>(index, any, value));
576             }
577             else
578             {
579                 if (!Exists(index))
580                 {
581                     return @continue;
582                 }
583                 if (AreEqual(value, any))
584                 {
585                     return handler(GetLinkStruct(index));
586                 }
587                 ref var storedLinkValue = ref GetLinkDataPartReference(index);
588                 if (AreEqual(storedLinkValue.Source, value) ||
589                     AreEqual(storedLinkValue.Target, value))
590                 {
591                     return handler(GetLinkStruct(index));
592                 }
593                 return @continue;
594             }
595         }
596     }

```

```

595 }
596 if (restrictions.Count == 3)
597 {
598     var externalReferencesRange = constants.ExternalReferencesRange;
599     var source = restrictions[constants.SourcePart];
600     var target = restrictions[constants.TargetPart];
601     if (AreEqual(index, any))
602     {
603         if (AreEqual(source, any) && AreEqual(target, any))
604         {
605             return Each(handler, Array.Empty<TLink>());
606         }
607         else if (AreEqual(source, any))
608         {
609             if (externalReferencesRange.HasValue &&
610                 ↪ externalReferencesRange.Value.Contains(target))
611             {
612                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
613             }
614             else
615             {
616                 return InternalTargetsTreeMethods.EachUsage(target, handler);
617             }
618         }
619         else if (AreEqual(target, any))
620         {
621             if (externalReferencesRange.HasValue &&
622                 ↪ externalReferencesRange.Value.Contains(source))
623             {
624                 return ExternalSourcesTreeMethods.EachUsage(source, handler);
625             }
626             else
627             {
628                 if (_useLinkedList)
629                 {
630                     return InternalSourcesListMethods.EachUsage(source, handler);
631                 }
632                 else
633                 {
634                     return InternalSourcesTreeMethods.EachUsage(source, handler);
635                 }
636             }
637         }
638         else //if(source != Any && target != Any)
639         {
640             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             }
664             else
665             {
666                 if (_useLinkedList ||
667                     ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
668                     ↪ InternalTargetsTreeMethods.CountUsages(target)))
669                 {
670                     link = InternalTargetsTreeMethods.Search(source, target);
671                 }
672                 else
673                 {

```

```

668         link = InternalSourcesTreeMethods.Search(source, target);
669     }
670 }
671 }
672 else
673 {
674     if (_useLinkedList ||
        ↳ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
        ↳ InternalTargetsTreeMethods.CountUsages(target)))
        {
675         link = InternalTargetsTreeMethods.Search(source, target);
676     }
677     else
678     {
679         link = InternalSourcesTreeMethods.Search(source, target);
680     }
681 }
682 return AreEqual(link, constants.Null) ? @continue :
683     ↳ 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(IList<TLink> restrictions, IList<TLink> substitution)
732 {
733     var constants = Constants;
734     var @null = constants.Null;
735     var externalReferencesRange = constants.ExternalReferencesRange;
736     var linkIndex = restrictions[constants.IndexPart];
737     ref var link = ref GetLinkDataPartReference(linkIndex);
738     var source = link.Source;
739     var target = link.Target;
740     ref var header = ref GetHeaderReference();
741     ref var rootAsSource = ref header.RootAsSource;

```

```

741     ref var rootAsTarget = ref header.RootAsTarget;
742     // Будет корректно работать только в том случае, если пространство выделенной связи
743     ↪ предварительно заполнено нулями
744     if (!AreEqual(source, @null))
745     {
746         if (externalReferencesRange.HasValue &&
747             ↪ externalReferencesRange.Value.Contains(source))
748         {
749             ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
750         }
751         else
752         {
753             if (_useLinkedList)
754             {
755                 InternalSourcesListMethods.Detach(source, linkIndex);
756             }
757             else
758             {
759                 InternalSourcesTreeMethods.Detach(ref
760                     ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
761             }
762         }
763     }
764     if (!AreEqual(target, @null))
765     {
766         if (externalReferencesRange.HasValue &&
767             ↪ externalReferencesRange.Value.Contains(target))
768         {
769             ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
770         }
771         else
772         {
773             InternalTargetsTreeMethods.Detach(ref
774                 ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
775         }
776     }
777     source = link.Source = substitution[constants.SourcePart];
778     target = link.Target = substitution[constants.TargetPart];
779     if (!AreEqual(source, @null))
780     {
781         if (externalReferencesRange.HasValue &&
782             ↪ externalReferencesRange.Value.Contains(source))
783         {
784             ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
785         }
786         else
787         {
788             if (_useLinkedList)
789             {
790                 InternalSourcesListMethods.AttachAsLast(source, linkIndex);
791             }
792             else
793             {
794                 InternalSourcesTreeMethods.Attach(ref
795                     ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
796             }
797         }
798     }
799     if (!AreEqual(target, @null))
800     {
801         if (externalReferencesRange.HasValue &&
802             ↪ externalReferencesRange.Value.Contains(target))
803         {
804             ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
805         }
806         else
807         {
808             InternalTargetsTreeMethods.Attach(ref
809                 ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
810         }
811     }
812     return linkIndex;
813 }
814
815 /// <remarks>
816 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
817 ↪ пространство
818 /// </remarks>

```

```

809 [MethodImpl(MethodImplOptions.AggressiveInlining)]
810 public virtual TLink Create(ICollection<TLink> restrictions)
811 {
812     ref var header = ref GetHeaderReference();
813     var freeLink = header.FirstFreeLink;
814     if (!AreEqual(freeLink, Constants.Null))
815     {
816         UnusedLinksListMethods.Detach(freeLink);
817     }
818     else
819     {
820         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
821         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
822         {
823             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
824         }
825         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
826         {
827             _dataMemory.ReservedCapacity += _dataMemory.ReservationStepInBytes;
828             _indexMemory.ReservedCapacity += _indexMemory.ReservationStepInBytes;
829             SetPointers(_dataMemory, _indexMemory);
830             header = ref GetHeaderReference();
831             header.ReservedLinks = ConvertToAddress(_dataMemory.ReservedCapacity /
832                 ↳ LinkDataPartSizeInBytes);
833         }
834         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
835         _dataMemory.UsedCapacity += LinkDataPartSizeInBytes;
836         _indexMemory.UsedCapacity += LinkIndexPartSizeInBytes;
837     }
838     return freeLink;
839 }
840 /// <summary>
841 /// <para>
842 /// Deletes the restrictions.
843 /// </para>
844 /// <para></para>
845 /// </summary>
846 /// <param name="restrictions">
847 /// <para>The restrictions.</para>
848 /// <para></para>
849 /// </param>
850 [MethodImpl(MethodImplOptions.AggressiveInlining)]
851 public virtual void Delete(ICollection<TLink> restrictions)
852 {
853     ref var header = ref GetHeaderReference();
854     var link = restrictions[Constants.IndexPart];
855     if (LessThan(link, header.AllocatedLinks)
856     {
857         UnusedLinksListMethods.AttachAsFirst(link);
858     }
859     else if (AreEqual(link, header.AllocatedLinks))
860     {
861         header.AllocatedLinks = Decrement(header.AllocatedLinks);
862         _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
863         _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
864         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
865         ↳ пока не дойдём до первой существующей связи
866         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
867         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
868             ↳ IsUnusedLink(header.AllocatedLinks))
869         {
870             UnusedLinksListMethods.Detach(header.AllocatedLinks);
871             header.AllocatedLinks = Decrement(header.AllocatedLinks);
872             _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
873             _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
874         }
875     }
876 }
877 /// <summary>
878 /// <para>
879 /// Gets the link struct using the specified link index.
880 /// </para>
881 /// <para></para>
882 /// </summary>
883 /// <param name="linkIndex">
884 /// <para>The link index.</para>
885 /// <para></para>

```



```

885 /// </param>
886 /// <returns>
887 /// <para>A list of t link</para>
888 /// <para></para>
889 /// </returns>
890 [MethodImpl(MethodImplOptions.AggressiveInlining)]
891 public IList<TLink> GetLinkStruct(TLink linkIndex)
892 {
893     ref var link = ref GetLinkDataPartReference(linkIndex);
894     return new Link<TLink>(linkIndex, link.Source, link.Target);
895 }
896
897 /// <remarks>
898 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
899   ↪ адрес реально поменялся
900 ///
901 /// Указатель this.links может быть в том же месте,
902 /// так как 0-я связь не используется и имеет такой же размер как Header,
903 /// поэтому header размещается в том же месте, что и 0-я связь
904 /// </remarks>
905 [MethodImpl(MethodImplOptions.AggressiveInlining)]
906 protected abstract void SetPointers(IResizableDirectMemory dataMemory,
907   ↪ IResizableDirectMemory indexMemory);
908
909 /// <summary>
910 /// <para>
911 /// Resets the pointers.
912 /// </para>
913 /// <para></para>
914 /// </summary>
915 [MethodImpl(MethodImplOptions.AggressiveInlining)]
916 protected virtual void ResetPointers()
917 {
918     InternalSourcesListMethods = null;
919     InternalSourcesTreeMethods = null;
920     ExternalSourcesTreeMethods = null;
921     InternalTargetsTreeMethods = null;
922     ExternalTargetsTreeMethods = null;
923     UnusedLinksListMethods = null;
924 }
925
926 /// <summary>
927 /// <para>
928 /// Gets the header reference.
929 /// </para>
930 /// <para></para>
931 /// </summary>
932 /// <returns>
933 /// <para>A ref links header of t link</para>
934 /// <para></para>
935 /// </returns>
936 [MethodImpl(MethodImplOptions.AggressiveInlining)]
937 protected abstract ref LinksHeader<TLink> GetHeaderReference();
938
939 /// <summary>
940 /// <para>
941 /// Gets the link data part reference using the specified link index.
942 /// </para>
943 /// <para></para>
944 /// </summary>
945 /// <param name="linkIndex">
946 /// <para>The link index.</para>
947 /// <para></para>
948 /// </param>
949 /// <returns>
950 /// <para>A ref raw link data part of t link</para>
951 /// <para></para>
952 /// </returns>
953 [MethodImpl(MethodImplOptions.AggressiveInlining)]
954 protected abstract ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex);
955
956 /// <summary>
957 /// <para>
958 /// Gets the link index part reference using the specified link index.
959 /// </para>
960 /// <para></para>
961 /// </summary>
962 /// <param name="linkIndex">

```

```

961     /// <para>The link index.</para>
962     /// <para></para>
963     /// </param>
964     /// <returns>
965     /// <para>A ref raw link index part of t link</para>
966     /// <para></para>
967     /// </returns>
968     [MethodImpl(MethodImplOptions.AggressiveInlining)]
969     protected abstract ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
    ↪ linkIndex);

970
971     /// <summary>
972     /// <para>
973     /// Determines whether this instance exists.
974     /// </para>
975     /// <para></para>
976     /// </summary>
977     /// <param name="link">
978     /// <para>The link.</para>
979     /// <para></para>
980     /// </param>
981     /// <returns>
982     /// <para>The bool</para>
983     /// <para></para>
984     /// </returns>
985     [MethodImpl(MethodImplOptions.AggressiveInlining)]
986     protected virtual bool Exists(TLink link)
987         => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
988         && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
989         && !IsUnusedLink(link);
990
991     /// <summary>
992     /// <para>
993     /// Determines whether this instance is unused link.
994     /// </para>
995     /// <para></para>
996     /// </summary>
997     /// <param name="linkIndex">
998     /// <para>The link index.</para>
999     /// <para></para>
1000    /// </param>
1001    /// <returns>
1002    /// <para>The bool</para>
1003    /// <para></para>
1004    /// </returns>
1005    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1006    protected virtual bool IsUnusedLink(TLink linkIndex)
1007    {
1008        if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
1009            ↪ is not needed
1010        {
1011            // TODO: Reduce access to memory in different location (should be enough to use
1012            ↪ just linkIndexPart)
1013            ref var linkDataPart = ref GetLinkDataPartReference(linkIndex);
1014            ref var linkIndexPart = ref GetLinkIndexPartReference(linkIndex);
1015            return AreEqual(linkIndexPart.SizeAsTarget, default) &&
1016                ↪ !AreEqual(linkDataPart.Source, default);
1017        }
1018        else
1019        {
1020            return true;
1021        }
1022    }
1023
1024    /// <summary>
1025    /// <para>
1026    /// Gets the one.
1027    /// </para>
1028    /// <para></para>
1029    /// </summary>
1030    /// <returns>
1031    /// <para>The link</para>
1032    /// <para></para>
1033    /// </returns>
1034    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1035    protected virtual TLink GetOne() => _one;

```

```

1035     /// <para>
1036     /// Gets the zero.
1037     /// </para>
1038     /// <para></para>
1039     /// </summary>
1040     /// <returns>
1041     /// <para>The link</para>
1042     /// <para></para>
1043     /// </returns>
1044     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1045     protected virtual TLink GetZero() => default;
1046
1047     /// <summary>
1048     /// <para>
1049     /// Determines whether this instance are equal.
1050     /// </para>
1051     /// <para></para>
1052     /// </summary>
1053     /// <param name="first">
1054     /// <para>The first.</para>
1055     /// <para></para>
1056     /// </param>
1057     /// <param name="second">
1058     /// <para>The second.</para>
1059     /// <para></para>
1060     /// </param>
1061     /// <returns>
1062     /// <para>The bool</para>
1063     /// <para></para>
1064     /// </returns>
1065     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1066     protected virtual bool AreEqual(TLink first, TLink second) =>
1067         ↪ _equalityComparer.Equals(first, second);
1068
1069     /// <summary>
1070     /// <para>
1071     /// Determines whether this instance less than.
1072     /// </para>
1073     /// <para></para>
1074     /// </summary>
1075     /// <param name="first">
1076     /// <para>The first.</para>
1077     /// <para></para>
1078     /// </param>
1079     /// <param name="second">
1080     /// <para>The second.</para>
1081     /// <para></para>
1082     /// </param>
1083     /// <returns>
1084     /// <para>The bool</para>
1085     /// <para></para>
1086     /// </returns>
1087     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1088     protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
1089         ↪ second) < 0;
1090
1091     /// <summary>
1092     /// <para>
1093     /// Determines whether this instance less or equal than.
1094     /// </para>
1095     /// <para></para>
1096     /// </summary>
1097     /// <param name="first">
1098     /// <para>The first.</para>
1099     /// <para></para>
1100     /// </param>
1101     /// <param name="second">
1102     /// <para>The second.</para>
1103     /// <para></para>
1104     /// </param>
1105     /// <returns>
1106     /// <para>The bool</para>
1107     /// <para></para>
1108     /// </returns>
1109     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1110     protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
1111         ↪ _comparer.Compare(first, second) <= 0;

```

```

1110    /// <summary>
1111    /// <para>
1112    /// Determines whether this instance greater than.
1113    /// </para>
1114    /// <para></para>
1115    /// </summary>
1116    /// <param name="first">
1117    /// <para>The first.</para>
1118    /// <para></para>
1119    /// </param>
1120    /// <param name="second">
1121    /// <para>The second.</para>
1122    /// <para></para>
1123    /// </param>
1124    /// <returns>
1125    /// <para>The bool</para>
1126    /// <para></para>
1127    /// </returns>
1128    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1129    protected virtual bool GreaterThan(TLink first, TLink second) =>
1130        ↪ _comparer.Compare(first, second) > 0;
1131
1132    /// <summary>
1133    /// <para>
1134    /// Determines whether this instance greater or equal than.
1135    /// </para>
1136    /// <para></para>
1137    /// </summary>
1138    /// <param name="first">
1139    /// <para>The first.</para>
1140    /// <para></para>
1141    /// </param>
1142    /// <param name="second">
1143    /// <para>The second.</para>
1144    /// <para></para>
1145    /// </param>
1146    /// <returns>
1147    /// <para>The bool</para>
1148    /// <para></para>
1149    /// </returns>
1150    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1151    protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
1152        ↪ _comparer.Compare(first, second) >= 0;
1153
1154    /// <summary>
1155    /// <para>
1156    /// Converts the to int 64 using the specified value.
1157    /// </para>
1158    /// <para></para>
1159    /// </summary>
1160    /// <param name="value">
1161    /// <para>The value.</para>
1162    /// <para></para>
1163    /// </param>
1164    /// <returns>
1165    /// <para>The long</para>
1166    /// <para></para>
1167    /// </returns>
1168    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1169    protected virtual long ConvertToInt64(TLink value) =>
1170        ↪ _addressToInt64Converter.Convert(value);
1171
1172    /// <summary>
1173    /// <para>
1174    /// Converts the to address using the specified value.
1175    /// </para>
1176    /// <para></para>
1177    /// </summary>
1178    /// <param name="value">
1179    /// <para>The value.</para>
1180    /// <para></para>
1181    /// </param>
1182    /// <returns>
1183    /// <para>The link</para>
1184    /// <para></para>
1185    /// </returns>
1186    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

1184     protected virtual TLink ConvertToAddress(long value) =>
1185         ↪ _int64ToAddressConverter.Convert(value);
1186
1187     /// <summary>
1188     /// <para>
1189     /// Adds the first.
1190     /// </para>
1191     /// </summary>
1192     /// <param name="first">
1193     /// <para>The first.</para>
1194     /// <para></para>
1195     /// </param>
1196     /// <param name="second">
1197     /// <para>The second.</para>
1198     /// <para></para>
1199     /// </param>
1200     /// <returns>
1201     /// <para>The link</para>
1202     /// <para></para>
1203     /// </returns>
1204     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1205     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
1206         ↪ second);
1207
1208     /// <summary>
1209     /// <para>
1210     /// Subtracts the first.
1211     /// </para>
1212     /// </summary>
1213     /// <param name="first">
1214     /// <para>The first.</para>
1215     /// <para></para>
1216     /// </param>
1217     /// <param name="second">
1218     /// <para>The second.</para>
1219     /// <para></para>
1220     /// </param>
1221     /// <returns>
1222     /// <para>The link</para>
1223     /// <para></para>
1224     /// </returns>
1225     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1226     protected virtual TLink Subtract(TLink first, TLink second) =>
1227         ↪ Arithmetic<TLink>.Subtract(first, second);
1228
1229     /// <summary>
1230     /// <para>
1231     /// Increments the link.
1232     /// </para>
1233     /// </summary>
1234     /// <param name="link">
1235     /// <para>The link.</para>
1236     /// <para></para>
1237     /// </param>
1238     /// <returns>
1239     /// <para>The link</para>
1240     /// <para></para>
1241     /// </returns>
1242     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1243     protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
1244
1245     /// <summary>
1246     /// <para>
1247     /// Decrements the link.
1248     /// </para>
1249     /// <para></para>
1250     /// </summary>
1251     /// <param name="link">
1252     /// <para>The link.</para>
1253     /// <para></para>
1254     /// </param>
1255     /// <returns>
1256     /// <para>The link</para>
1257     /// <para></para>
1258     /// </returns>

```

```

1259     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1260     protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
1261
1262     #region Disposable
1263
1264     /// <summary>
1265     /// <para>
1266     /// Gets the allow multiple dispose calls value.
1267     /// </para>
1268     /// <para></para>
1269     /// </summary>
1270     protected override bool AllowMultipleDisposeCalls
1271     {
1272         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1273         get => true;
1274     }
1275
1276     /// <summary>
1277     /// <para>
1278     /// Disposes the manual.
1279     /// </para>
1280     /// <para></para>
1281     /// </summary>
1282     /// <param name="manual">
1283     /// <para>The manual.</para>
1284     /// <para></para>
1285     /// </param>
1286     /// <param name="wasDisposed">
1287     /// <para>The was disposed.</para>
1288     /// <para></para>
1289     /// </param>
1290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1291     protected override void Dispose(bool manual, bool wasDisposed)
1292     {
1293         if (!wasDisposed)
1294         {
1295             ResetPointers();
1296             _dataMemory.DisposeIfPossible();
1297             _indexMemory.DisposeIfPossible();
1298         }
1299     }
1300
1301     #endregion
1302 }
1303 }

```

1.44 ./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
24         private readonly byte* _links;
25         private readonly byte* _header;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="UnusedLinksListMethods"> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>

```

```

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

```

```

106    /// <para></para>
107    /// </summary>
108    /// <param name="element">
109    /// <para>The element.</para>
110    /// <para></para>
111    /// </param>
112    /// <returns>
113    /// <para>The link</para>
114    /// <para></para>
115    /// </returns>
116    [MethodImpl(MethodImplOptions.AggressiveInlining)]
117    protected override TLink GetPrevious(TLink element) =>
118        ↪ GetLinkDataPartReference(element).Source;
119
120    /// <summary>
121    /// <para>
122    /// Gets the next using the specified element.
123    /// </para>
124    /// <para></para>
125    /// </summary>
126    /// <param name="element">
127    /// <para>The element.</para>
128    /// <para></para>
129    /// </param>
130    /// <returns>
131    /// <para>The link</para>
132    /// <para></para>
133    /// </returns>
134    [MethodImpl(MethodImplOptions.AggressiveInlining)]
135    protected override TLink GetNext(TLink element) =>
136        ↪ GetLinkDataPartReference(element).Target;
137
138    /// <summary>
139    /// <para>
140    /// Gets the size.
141    /// </para>
142    /// <para></para>
143    /// </summary>
144    /// <returns>
145    /// <para>The link</para>
146    /// <para></para>
147    /// </returns>
148    [MethodImpl(MethodImplOptions.AggressiveInlining)]
149    protected override TLink GetSize() => GetHeaderReference().FreeLinks;
150
151    /// <summary>
152    /// <para>
153    /// Sets the first using the specified element.
154    /// </para>
155    /// <para></para>
156    /// </summary>
157    /// <param name="element">
158    /// <para>The element.</para>
159    /// <para></para>
160    /// </param>
161    [MethodImpl(MethodImplOptions.AggressiveInlining)]
162    protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
163        ↪ element;
164
165    /// <summary>
166    /// <para>
167    /// Sets the last using the specified element.
168    /// </para>
169    /// <para></para>
170    /// </summary>
171    /// <param name="element">
172    /// <para>The element.</para>
173    /// <para></para>
174    /// </param>
175    [MethodImpl(MethodImplOptions.AggressiveInlining)]
176    protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
177        ↪ element;
178
179    /// <summary>
180    /// <para>
181    /// Sets the previous using the specified element.
182    /// </para>
183    /// <para></para>
184    /// </summary>
185    /// <param name="element">
186    /// <para>The element.</para>
187    /// <para></para>
188    /// </param>
189    [MethodImpl(MethodImplOptions.AggressiveInlining)]
190    protected override void SetPrevious(TLink element) => GetHeaderReference().PreviousFreeLink =
191        ↪ element;
192
193    /// <summary>
194    /// <para>
195    /// Sets the next using the specified element.
196    /// </para>
197    /// <para></para>
198    /// </summary>
199    /// <param name="element">
200    /// <para>The element.</para>
201    /// <para></para>
202    /// </param>
203    [MethodImpl(MethodImplOptions.AggressiveInlining)]
204    protected override void SetNext(TLink element) => GetHeaderReference().NextFreeLink =
205        ↪ element;

```



```

180     /// </summary>
181     /// <param name="element">
182     /// <para>The element.</para>
183     /// <para></para>
184     /// </param>
185     /// <param name="previous">
186     /// <para>The previous.</para>
187     /// <para></para>
188     /// </param>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override void SetPrevious(TLink element, TLink previous) =>
191         ↪ GetLinkDataPartReference(element).Source = previous;
192
193     /// <summary>
194     /// <para>
195     /// Sets the next using the specified element.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     /// <param name="element">
200     /// <para>The element.</para>
201     /// <para></para>
202     /// </param>
203     /// <param name="next">
204     /// <para>The next.</para>
205     /// <para></para>
206     /// </param>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override void SetNext(TLink element, TLink next) =>
209         ↪ GetLinkDataPartReference(element).Target = next;
210
211     /// <summary>
212     /// <para>
213     /// Sets the size using the specified size.
214     /// </para>
215     /// <para></para>
216     /// </summary>
217     /// <param name="size">
218     /// <para>The size.</para>
219     /// <para></para>
220     /// </param>
221     [MethodImpl(MethodImplOptions.AggressiveInlining)]
222     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
223 }

```

1.45 ./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>

```

```

32     /// <para></para>
33     /// </summary>
34     public TLink Source;
35     /// <summary>
36     /// <para>
37     /// The target.
38     /// </para>
39     /// <para></para>
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 }

```

1.46 ./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         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLinkIndexPart<TLink>>.Size;
28
29         /// <summary>
30         /// <para>
31         /// The root as source.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public TLink RootAsSource;
36         /// <summary>
37         /// <para>
38         /// The left as source.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         public TLink LeftAsSource;
43         /// <summary>
44         /// <para>
45         /// The right as source.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         public TLink RightAsSource;
50         /// <summary>
51         /// <para>
52         /// The size as source.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         public TLink SizeAsSource;
57         /// <summary>
58         /// <para>
59         /// The root as target.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         public TLink RootAsTarget;
64         /// <summary>
65         /// <para>
66         /// The left as target.
67         /// </para>
68         /// <para></para>
69         /// </summary>
70         public TLink LeftAsTarget;
71         /// <summary>
72         /// <para>
73         /// The right as target.
74         /// </para>
75         /// <para></para>
76         /// </summary>
77         public TLink RightAsTarget;
78         /// <summary>
79         /// <para>
80         /// The size as target.
81         /// </para>
82         /// <para></para>
83         /// </summary>
84         public TLink SizeAsTarget;

```

```

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 }

```

1.47 ./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;

```

```

84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equal to zero.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="value">
92     /// <para>The value.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100     protected override bool EqualToZero(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>

```

```

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

```

1.48 ./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 :
    ↳ ExternalLinksSizeBalancedTreeMethodsBase<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 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>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header)
66      : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
67      {
68          LinksDataParts = linksDataParts;
69          LinksIndexParts = linksIndexParts;
70          Header = header;
71      }
72
73      /// <summary>
74      /// <para>
75      /// Gets the zero.
76      /// </para>
77      /// <para></para>
78      /// </summary>
79      /// <returns>
80      /// <para>The 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>

```

```

89     /// <para></para>
90     /// </summary>
91     /// <param name="value">
92     /// <para>The value.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100     protected override bool EqualToZero(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)]

```

```

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

```

1.49 ./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     /// </summary>
13     /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
15         ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>

```



```

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

```

```

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.
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 FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
236         ↳ TLink secondSource, TLink secondTarget)
237         ↳ => firstSource < secondSource || firstSource == secondSource && firstTarget <
238         ↳ secondTarget;
239
240     /// <summary>
241     /// <para>

```

```

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

```

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

```

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     }
30 }

```

```

25     /// <para></para>
26     /// </param>
27     /// <param name="linksDataParts">
28     /// <para>A links data parts.</para>
29     /// <para></para>
30     /// </param>
31     /// <param name="linksIndexParts">
32     /// <para>A links index parts.</para>
33     /// <para></para>
34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt32 ExternalLinksSourcesSizeBalancedTreeMethods(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>

```

```

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 }

```

1.51 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethodsBase

```

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>

```



```

34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public
41    ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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].LeftAsTarget;
63
64    /// <summary>
65    /// <para>
66    /// Gets the right reference using the specified node.
67    /// </para>
68    /// <para></para>
69    /// </summary>
70    /// <param name="node">
71    /// <para>The node.</para>
72    /// <para></para>
73    /// </param>
74    /// <returns>
75    /// <para>The ref link</para>
76    /// <para></para>
77    /// </returns>
78    [MethodImpl(MethodImplOptions.AggressiveInlining)]
79    protected override ref TLink GetRightReference(TLink 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>

```

```

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.

```

```

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() => 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 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">

```

```

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         => 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     }
288 }
289 }

```

1.52 ./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)]

```

```

40 public UInt32ExternalLinksTargetsSizeBalancedTreeMethods(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>
190     /// </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 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>

```

```

264 [MethodImpl(MethodImplOptions.AggressiveInlining)]
265 protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
    ↳ TLink secondSource, TLink secondTarget)
266 => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
    ↳ secondSource;
267
268 /// <summary>
269 /// <para>
270 /// Clears the node using the specified node.
271 /// </para>
272 /// <para></para>
273 /// </summary>
274 /// <param name="node">
275 /// <para>The node.</para>
276 /// <para></para>
277 /// </param>
278 [MethodImpl(MethodImplOptions.AggressiveInlining)]
279 protected override void ClearNode(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 }

```

1.53 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase

```

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 :
    ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<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="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase" /> instance.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="constants">
47        /// <para>A constants.</para>
48        /// <para></para>
49        /// </param>

```



```

50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// </para>
53     /// </param>
54     /// <param name="linksIndexParts">
55     /// <para>A links index parts.</para>
56     /// </para>
57     /// </param>
58     /// <param name="header">
59     /// <para>A header.</para>
60     /// </para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected
64     ↪ UInt32 InternalLinksRecursionlessSizeBalancedTreeMethodsBase(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     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override TLink GetZero() => 0U;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equal to zero.
88     /// </para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// </para>
93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// </returns>
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     protected override bool EqualToZero(TLink value) => value == 0U;
99
100     /// <summary>
101     /// <para>
102     /// Determines whether this instance are equal.
103     /// </para>
104     /// </summary>
105     /// <param name="first">
106     /// <para>The first.</para>
107     /// </para>
108     /// </param>
109     /// <param name="second">
110     /// <para>The second.</para>
111     /// </para>
112     /// </param>
113     /// <returns>
114     /// <para>The bool</para>
115     /// </returns>
116     [MethodImpl(MethodImplOptions.AggressiveInlining)]
117     protected override bool AreEqual(TLink first, TLink second) => first == second;
118
119     /// <summary>
120     /// <para>
121     /// 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
197     ↪ always true for ulong
198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance less or equal than zero.
202     /// </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) =>
        ↪ 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) =>
        ↪ ref LinksIndexParts[link];
383
384     /// <summary>
385     /// <para>
386     /// Determines whether this instance first is to the left of second.
387     /// </para>
388     /// <para></para>
389     /// </summary>
390     /// <param name="first">
391     /// <para>The first.</para>
392     /// <para></para>
393     /// </param>
394     /// <param name="second">
395     /// <para>The second.</para>
396     /// <para></para>
397     /// </param>
398     /// <returns>
399     /// <para>The bool</para>
400     /// <para></para>
401     /// </returns>
402     [MethodImpl(MethodImplOptions.AggressiveInlining)]
403     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
        ↪ GetKeyPartValue(first) < GetKeyPartValue(second);
404
405     /// <summary>
406     /// <para>
407     /// Determines whether this instance first is to the right of second.
408     /// </para>
409     /// <para></para>
410     /// </summary>
411     /// <param name="first">
412     /// <para>The first.</para>
413     /// <para></para>
414     /// </param>
415     /// <param name="second">
416     /// <para>The second.</para>
417     /// <para></para>
418     /// </param>
419     /// <returns>
420     /// <para>The bool</para>
421     /// <para></para>
422     /// </returns>
423     [MethodImpl(MethodImplOptions.AggressiveInlining)]
424     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
        ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
425 }
426

```

1.54 ./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 :
17     ↪ InternalLinksSizeBalancedTreeMethodsBase<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 cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
44         ↪ instance.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="constants">
49         /// <para>A constants.</para>
50         /// <para></para>
51         /// </param>
52         /// <param name="linksDataParts">
53         /// <para>A links data parts.</para>
54         /// <para></para>
55         /// </param>
56         /// <param name="linksIndexParts">
57         /// <para>A links index parts.</para>
58         /// <para></para>
59         /// </param>
60         /// <param name="header">
61         /// <para>A header.</para>
62         /// <para></para>
63         /// </param>
64         [MethodImpl(MethodImplOptions.AggressiveInlining)]
65         protected UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<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>

```

```

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() => OU;
83
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance equal to zero.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="value">
91     /// <para>The value.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>The bool</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override bool EqualToZero(TLink value) => value == OU;
100
101     /// <summary>
102     /// <para>
103     /// Determines whether this instance are equal.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="first">
108     /// <para>The first.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="second">
112     /// <para>The second.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The bool</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override bool AreEqual(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 > OU;
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
197     ↪ always true for ulong
198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance less or equal than zero.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="value">
206     /// <para>The value.</para>
207     /// <para></para>
208     /// </param>
209     /// <returns>
210     /// <para>The bool</para>
211     /// <para></para>
212     /// </returns>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     protected override bool LessOrEqualThanZero(TLink value) => value == 0UL; // value is
215     ↪ always >= 0 for ulong
216
217     /// <summary>
218     /// <para>
219     /// Determines whether this instance less or equal than.
220     /// </para>
221     /// <para></para>
222     /// </summary>
223     /// <param name="first">
224     /// <para>The first.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="second">
228     /// <para>The second.</para>
229     /// <para></para>
230     /// </param>

```



```

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)]

```

```

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 }

```

1.55 ./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     /// </summary>
13     /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLink}"/>
14     public unsafe class UInt32InternalLinksSourcesLinkedListMethods :
15         ↪ 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="UInt32InternalLinksSourcesLinkedListMethods"/> instance.
23         /// </para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// </param>

```

```

28     /// <para></para>
29     /// </param>
30     /// <param name="linksDataParts">
31     /// <para>A links data parts.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="linksIndexParts">
35     /// <para>A links index parts.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<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     /// <summary>
47     /// <para>
48     /// Gets the link data part reference using the specified link.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="link">
53     /// <para>The link.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>A ref raw link data part of t link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
62     ↪ ref _linksDataParts[link];
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) =>
79     ↪ ref _linksIndexParts[link];
80 }

```

1.56 ./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>

```

```

19  /// Initializes a new <see
    ↪ cref="UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
20  /// </para>
21  /// <para></para>
22  /// </summary>
23  /// <param name="constants">
24  /// <para>A constants.</para>
25  /// <para></para>
26  /// </param>
27  /// <param name="linksDataParts">
28  /// <para>A links data parts.</para>
29  /// <para></para>
30  /// </param>
31  /// <param name="linksIndexParts">
32  /// <para>A links index parts.</para>
33  /// <para></para>
34  /// </param>
35  /// <param name="header">
36  /// <para>A header.</para>
37  /// <para></para>
38  /// </param>
39  [MethodImpl(MethodImplOptions.AggressiveInlining)]
40  public
    ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<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>

```

```

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>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;
229
230     /// <summary>
231     /// <para>
232     /// Clears the node using the specified node.
233     /// </para>
234     /// <para></para>
235     /// </summary>
236     /// <param name="node">
237     /// <para>The node.</para>
238     /// <para></para>
239     /// </param>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     protected override void ClearNode(TLink node)
242     {
243         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 }

```

1.57 ./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) { }
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     /// </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 using the specified node.
175     /// </para>
176     /// </summary>
177     /// <param name="node">
178     /// <para>The node.</para>
179     /// </param>
180     /// <returns>
181     /// <para>The link</para>
182     /// </returns>
183     [MethodImpl(MethodImplOptions.AggressiveInlining)]
184     protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsSource;
185
186     /// <summary>
187     /// <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);
268 }

```

1.58 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalance

```

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 :
16    ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see
21        ↪ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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        ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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>
61        /// </returns>
62        [MethodImpl(MethodImplOptions.AggressiveInlining)]
63        protected override ref TLink GetLeftReference(TLink node) => ref
64        ↪ LinksIndexParts[node].LeftAsTarget;
65
66        /// <summary>
67        /// <para>
68        /// Gets the right reference using the specified node.
69        /// </para>
70        /// <para></para>
71        /// </summary>
72        /// <param name="node">
73        /// <para>The node.</para>
74        /// <para></para>
75        /// </param>
76        /// <returns>
77        /// <para>The ref link</para>
78        /// <para></para>
79        /// </returns>
80        [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     /// </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>

```

```

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].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 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>

```

```

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 }

```

1.59 ./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 :
    ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt32InternalLinksTargetsSizeBalancedTreeMethods"/>
    ↪ instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="linksDataParts">
28        /// <para>A links data parts.</para>
29        /// <para></para>
30        /// </param>

```

```

31    /// <param name="linksIndexParts">
32    /// <para>A links index parts.</para>
33    /// <para></para>
34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public UInt32InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<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>

```

```

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].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 }

```

1.60 ./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
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46         /// <summary>
47         /// <para>
48         /// Initializes a new <see cref="UInt32SplitMemoryLinks" /> instance.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <param name="dataMemory">
53         /// <para>A data memory.</para>
54         /// <para></para>
55         /// </param>
56         /// <param name="indexMemory">
57         /// <para>A index memory.</para>
58         /// <para></para>
59         /// </param>
60         /// <param name="memoryReservationStep">
61         /// <para>A memory reservation step.</para>
62         /// <para></para>
63         /// </param>

```

```

64 [MethodImpl(MethodImplOptions.AggressiveInlining)]
65 public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
    ↳ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
    ↳ IndexTreeType.Default, useLinkedList: true) { }

66
67 /// <summary>
68 /// <para>
69 /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
70 /// </para>
71 /// </para></para>
72 /// </summary>
73 /// <param name="dataMemory">
74 /// <para>A data memory.</para>
75 /// </para></para>
76 /// </param>
77 /// <param name="indexMemory">
78 /// <para>A index memory.</para>
79 /// </para></para>
80 /// </param>
81 /// <param name="memoryReservationStep">
82 /// <para>A memory reservation step.</para>
83 /// </para></para>
84 /// </param>
85 /// <param name="constants">
86 /// <para>A constants.</para>
87 /// </para></para>
88 /// </param>
89 [MethodImpl(MethodImplOptions.AggressiveInlining)]
90 public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep, LinksConstants<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
130         ↪ UInt32InternalLinksTargetsSizeBalancedTreeMethods(Constants,
131         ↪ _linksDataParts, _linksIndexParts, _header);
132     _createExternalTargetTreeMethods = () => new
133     ↪ UInt32ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
134     ↪ _linksDataParts, _linksIndexParts, _header);
135 }
136 else
137 {
138     _createInternalSourceTreeMethods = () => new
139     ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
140     ↪ _linksDataParts, _linksIndexParts, _header);
141     _createExternalSourceTreeMethods = () => new
142     ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
143     ↪ _linksDataParts, _linksIndexParts, _header);
144     _createInternalTargetTreeMethods = () => new
145     ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
146     ↪ _linksDataParts, _linksIndexParts, _header);
147     _createExternalTargetTreeMethods = () => new
148     ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
149     ↪ _linksDataParts, _linksIndexParts, _header);
150 }
151 Init(dataMemory, indexMemory);
152 }
153
154 /// <summary>
155 /// <para>
156 /// Sets the pointers using the specified data memory.
157 /// </para>
158 /// <para></para>
159 /// </summary>
160 /// <param name="dataMemory">
161 /// <para>The data memory.</para>
162 /// <para></para>
163 /// </param>
164 /// <param name="indexMemory">
165 /// <para>The index memory.</para>
166 /// <para></para>
167 /// </param>
168 [MethodImpl(MethodImplOptions.AggressiveInlining)]
169 protected override void SetPointers(IResizableDirectMemory dataMemory,
170 ↪ IResizableDirectMemory indexMemory)
171 {
172     _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
173     _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
174     _header = (LinksHeader<TLink>*)indexMemory.Pointer;
175     if (_useLinkedList)
176     {
177         InternalSourcesListMethods = new
178         ↪ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
179         ↪ _linksIndexParts);
180     }
181     else
182     {
183         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
184     }
185     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
186     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
187     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
188     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
189 }
190
191 /// <summary>
192 /// <para>
193 /// Resets the pointers.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 [MethodImpl(MethodImplOptions.AggressiveInlining)]
198 protected override void ResetPointers()
199 {
200     base.ResetPointers();
201     _linksDataParts = null;
202     _linksIndexParts = null;
203     _header = null;
204 }
205
206 /// <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>

```

```

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(TLink first, TLink 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(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() => 0U;
355
356    /// <summary>
357    /// <para>
358    /// Gets the one.
359    /// </para>
360    /// <para></para>
361    /// </summary>
362    /// <returns>
363    /// <para>The link</para>
364    /// <para></para>
365    /// </returns>
366    [MethodImpl(MethodImplOptions.AggressiveInlining)]
367    protected override 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 }

```

1.61 ./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     /// </summary>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// </param>
29     /// <param name="header">
30     /// <para>A header.</para>
31     /// </param>
32     [MethodImpl(MethodImplOptions.AggressiveInlining)]
33     public UInt32UnusedLinksListMethods(RawLinkDataPart<TLink>* links, LinksHeader<TLink>*
34     ↪ header)
35     : base((byte*)links, (byte*)header)
36     {
37         _links = links;
38         _header = header;
39     }
40
41     /// <summary>
42     /// <para>
43     /// Gets the link data part reference using the specified link.
44     /// </para>
45     /// </summary>
46     /// <param name="link">
47     /// <para>The link.</para>
48     /// </param>
49     /// <returns>
50     /// <para>A ref raw link data part of t link</para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
54     ↪ ref _links[link];
55
56     /// <summary>
57     /// <para>
58     /// Gets the header reference.
59     /// </para>
60     /// </summary>
61     /// <returns>
62     /// <para>A ref links header of t link</para>
63     /// </returns>
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
66 }
67
68 }
69
70 }
71
72 }
73

```

1.62 ./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     /// </summary>
14     /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
15     /// <seealso cref="ILinksTreeMethods{TLink}"/>
16     public unsafe abstract class UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17     ↪ 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>

```

```

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

```

```

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

```

```

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

```

```

327 [MethodImpl(MethodImplOptions.AggressiveInlining)]
328 protected override ulong Add(ulong first, ulong second) => first + second;
329
330 /// <summary>
331 /// <para>
332 /// Subtracts the first.
333 /// </para>
334 /// <para></para>
335 /// </summary>
336 /// <param name="first">
337 /// <para>The first.</para>
338 /// <para></para>
339 /// </param>
340 /// <param name="second">
341 /// <para>The second.</para>
342 /// <para></para>
343 /// </param>
344 /// <returns>
345 /// <para>The ulong</para>
346 /// <para></para>
347 /// </returns>
348 [MethodImpl(MethodImplOptions.AggressiveInlining)]
349 protected override ulong Subtract(ulong first, ulong second) => first - second;
350
351 /// <summary>
352 /// <para>
353 /// Gets the header reference.
354 /// </para>
355 /// <para></para>
356 /// </summary>
357 /// <returns>
358 /// <para>A ref links header of t link</para>
359 /// <para></para>
360 /// </returns>
361 [MethodImpl(MethodImplOptions.AggressiveInlining)]
362 protected override ref LinksHeader<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>

```

```

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 }

```

1.63 ./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     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16     /// <seealso cref="ILinksTreeMethods{TLink}"/>
17     public unsafe abstract class UInt64ExternalLinksSizeBalancedTreeMethodsBase :
18         ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
19     {
20         /// <summary>
21         /// <para>
22         /// The links data parts.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         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="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() => 0UL;
87
88    /// <summary>
89    /// <para>
90    /// Determines whether this instance equal to zero.
91    /// </para>
92    /// <para></para>
93    /// </summary>
94    /// <param name="value">
95    /// <para>The value.</para>
96    /// <para></para>
97    /// </param>
98    /// <returns>
99    /// <para>The bool</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override bool EqualToZero(ulong value) => value == 0UL;

```

```

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

```

```

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

```

```

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

```

```

332    /// Subtracts the first.
333    /// </para>
334    /// <para></para>
335    /// </summary>
336    /// <param name="first">
337    /// <para>The first.</para>
338    /// <para></para>
339    /// </param>
340    /// <param name="second">
341    /// <para>The second.</para>
342    /// <para></para>
343    /// </param>
344    /// <returns>
345    /// <para>The ulong</para>
346    /// <para></para>
347    /// </returns>
348    [MethodImpl(MethodImplOptions.AggressiveInlining)]
349    protected override ulong Subtract(ulong first, ulong second) => first - second;
350
351    /// <summary>
352    /// <para>
353    /// Gets the header reference.
354    /// </para>
355    /// <para></para>
356    /// </summary>
357    /// <returns>
358    /// <para>A ref links header of t link</para>
359    /// <para></para>
360    /// </returns>
361    [MethodImpl(MethodImplOptions.AggressiveInlining)]
362    protected override ref LinksHeader<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>

```

```

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 }

```

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

```

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>

```

```

30    /// </param>
31    /// <param name="linksIndexParts">
32    /// <para>A links index parts.</para>
33    /// <para></para>
34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public
    ↪ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<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     /// <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() => Header->RootAsSource;
189
190     /// <summary>
191     /// <para>
192     /// Gets the base part value using the specified node.
193     /// </para>
194     /// <para></para>
195     /// </summary>
196     /// <param name="node">
197     /// <para>The node.</para>
198     /// <para></para>
199     /// </param>
200     /// <returns>
201     /// <para>The link</para>
202     /// <para></para>
203     /// </returns>
204     [MethodImpl(MethodImplOptions.AggressiveInlining)]
205     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
206
207     /// <summary>
208     /// <para>
209     /// Determines whether this instance first is to the left of second.
210     /// </para>
211     /// <para></para>
212     /// </summary>
213     /// <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,
235     ↪ TLink secondSource, TLink secondTarget)
236     ↪ => firstSource < secondSource || firstSource == secondSource && firstTarget <
237     ↪ secondTarget;
238
239     /// <summary>
240     /// <para>
241     /// Determines whether this instance first is to the right of second.
242     /// </para>
243     /// <para></para>
244     /// </summary>
245     /// <param name="firstSource">
246     /// <para>The first source.</para>
247     /// <para></para>
248     /// </param>
249     /// <param name="firstTarget">
250     /// <para>The first target.</para>
251     /// <para></para>
252     /// </param>
253     /// <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 }

```

1.65 ./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>

```

```

38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt64ExternalLinksSourcesSizeBalancedTreeMethods(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>
190     /// </returns>

```

```

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>

```

```

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 }

```

1.66 ./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>

```

```

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

```

```

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>
190    /// </returns>
191    [MethodImpl(MethodImplOptions.AggressiveInlining)]
192    protected override TLink GetTreeRoot() => Header->RootAsTarget;
193
194    /// <summary>
195    /// <para>
196    /// Gets the base part value using the specified node.
197    /// </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 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>

```

```

269     /// <para>
270     /// Clears the node using the specified node.
271     /// </para>
272     /// <para></para>
273     /// </summary>
274     /// <param name="node">
275     /// <para>The node.</para>
276     /// <para></para>
277     /// </param>
278     [MethodImpl(MethodImplOptions.AggressiveInlining)]
279     protected override void ClearNode(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 }

```

1.67 ./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>

```

```

54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref TLink GetLeftReference(TLink node) => ref
58         ↳ LinksIndexParts[node].LeftAsTarget;
59
60     /// <summary>
61     /// <para>
62     /// Gets the right reference using the specified node.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="node">
67     /// <para>The node.</para>
68     /// </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     /// </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    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// <para></para>
105    /// </returns>
106    [MethodImpl(MethodImplOptions.AggressiveInlining)]
107    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
108
109    /// <summary>
110    /// <para>
111    /// Sets the left using the specified node.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="node">
116    /// <para>The node.</para>
117    /// </param>
118    /// <param name="left">
119    /// <para>The left.</para>
120    /// </param>
121    /// </summary>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    protected override void SetLeft(TLink node, TLink left) =>
124        ↳ LinksIndexParts[node].LeftAsTarget = left;
125
126    /// <summary>
127    /// <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>

```

```

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.
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)]

```

```

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 }

```

1.68 ./csharp/Platform.Data.Doublets.Memory.Split.Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase

```

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         /// <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
65         ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
66         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
67         ↪ linksIndexParts, LinksHeader<TLink>* header)

```

```

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

```

```

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

```

```

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>

```

```

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

```

1.69 ./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     /// <para></para>
14     /// </summary>
15     /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16     public unsafe abstract class UInt64InternalLinksSizeBalancedTreeMethodsBase :
        ↪ 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="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>

```

```

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

```

```

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

```

```

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

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

```

```

324 /// <para></para>
325 /// </returns>
326 [MethodImpl(MethodImplOptions.AggressiveInlining)]
327 protected override ulong Add(ulong first, ulong second) => first + second;
328
329 /// <summary>
330 /// <para>
331 /// Subtracts the first.
332 /// </para>
333 /// <para></para>
334 /// </summary>
335 /// <param name="first">
336 /// <para>The first.</para>
337 /// <para></para>
338 /// </param>
339 /// <param name="second">
340 /// <para>The second.</para>
341 /// <para></para>
342 /// </param>
343 /// <returns>
344 /// <para>The ulong</para>
345 /// <para></para>
346 /// </returns>
347 [MethodImpl(MethodImplOptions.AggressiveInlining)]
348 protected override ulong Subtract(ulong first, ulong second) => first - second;
349
350 /// <summary>
351 /// <para>
352 /// Gets the link data part reference using the specified link.
353 /// </para>
354 /// <para></para>
355 /// </summary>
356 /// <param name="link">
357 /// <para>The link.</para>
358 /// <para></para>
359 /// </param>
360 /// <returns>
361 /// <para>A ref raw link data part of t link</para>
362 /// <para></para>
363 /// </returns>
364 [MethodImpl(MethodImplOptions.AggressiveInlining)]
365 protected override ref RawLinkDataPart<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>

```



```

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     /// </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) =>
425         ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
426 }

```

1.70 ./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 :
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="UInt64InternalLinksSourcesLinkedListMethods"/> instance.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         /// <param name="constants">
28         /// <para>A constants.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksDataParts">
32         /// <para>A links data parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="linksIndexParts">
36         /// <para>A links index parts.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public UInt64InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
41             ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
42             : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
43         {
44             _linksDataParts = linksDataParts;
45             _linksIndexParts = linksIndexParts;
46         }
47     }

```

```

46     /// <summary>
47     /// <para>
48     /// Gets the link data part reference using the specified link.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     /// <param name="link">
53     /// <para>The link.</para>
54     /// <para></para>
55     /// </param>
56     /// <returns>
57     /// <para>A ref raw link data part of t link</para>
58     /// <para></para>
59     /// </returns>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     protected override ref RawLinkDataPart<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 }

```

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

```

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
20         ↪ cref="UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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>

```

```

38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public
41     ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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    /// <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].RightAsSource;

```

```

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">

```

```

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 }

```

1.72 ./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 :
        ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64InternalLinksSourcesSizeBalancedTreeMethods"/>
20         ↪ instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UInt64InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<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)]
58         protected override ref TLink GetLeftReference(TLink node) => ref
        ↪ 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
        ↪ LinksIndexParts[node].RightAsSource;
76     }
77 }

```

```

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].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>

```

```

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>
211 [MethodImpl(MethodImplOptions.AggressiveInlining)]
212 protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
213
214 /// <summary>
215 /// <para>
216 /// Gets the key part value using the specified node.
217 /// </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 }

```

1.73 ./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>

```

```

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
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

```

```

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>

```

```

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     /// </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].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 }

```

1.74 ./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) =>
    ↳ LinksIndexParts[node].RightAsTarget = right;

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

177
178 /// <summary>
179 /// <para>
180 /// Gets the tree root using the specified node.
181 /// </para>
182 /// <para></para>
183 /// </summary>
184 /// <param name="node">
185 /// <para>The node.</para>
186 /// <para></para>
187 /// </param>
188 /// <returns>
189 /// <para>The link</para>
190 /// <para></para>
191 /// </returns>
192 [MethodImpl(MethodImplOptions.AggressiveInlining)]
193 protected override 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) =>
267         ↪ SearchCore(GetTreeRoot(target), source);
268 }

```

1.75 ./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
45             ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
46
47         /// <summary>
48         /// <para>
49         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="dataMemory">
54         /// <para>A data memory.</para>
55         /// <para></para>
56         /// </param>
57         /// <param name="indexMemory">
58         /// <para>A index memory.</para>
59         /// <para></para>
60         /// </param>
61         /// <param name="memoryReservationStep">
62         /// <para>A memory reservation step.</para>
63         /// <para></para>
64         /// </param>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
67             ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
68             ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
69             ↪ IndexTreeType.Default, useLinkedList: true) { }
70
71         /// <summary>
72         /// <para>
73         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
74         /// </para>
75         /// <para></para>
76         /// </summary>
77         /// <param name="dataMemory">
78         /// <para>A data memory.</para>
79         /// <para></para>
80         /// </param>
81         /// <param name="indexMemory">
82         /// <para>A index memory.</para>
83         /// <para></para>
84         /// </param>
85         /// <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
138             ↳ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
139             ↳ _linksDataParts, _linksIndexParts, _header);
140     }
141     Init(dataMemory, indexMemory);
142 }
143
144 /// <summary>
145 /// <para>
146 /// Sets the pointers using the specified data memory.
147 /// </para>
148 /// </summary>
149 /// <param name="dataMemory">
150 /// <para>The data memory.</para>
151 /// </param>
152 /// <param name="indexMemory">
153 /// <para>The index memory.</para>
154 /// </param>
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 protected override void SetPointers(IResizableDirectMemory dataMemory,
157     ↳ 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
165             ↳ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
166             ↳ _linksIndexParts);
167     }
168     else
169     {
170         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
171     }
172     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
173     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
174     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
175     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_linksDataParts, _header);
176 }
177
178 /// <summary>
179 /// <para>
180 /// Resets the pointers.
181 /// </para>
182 /// </summary>
183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
184 protected override void ResetPointers()
185 {
186     base.ResetPointers();
187     _linksDataParts = null;
188     _linksIndexParts = null;
189     _header = null;
190 }
191
192 /// <summary>
193 /// <para>
194 /// Gets the header reference.
195 /// </para>
196 /// </summary>
197 /// <returns>
198 /// <para>A ref links header of t link</para>
199 /// </returns>
200 [MethodImpl(MethodImplOptions.AggressiveInlining)]
201 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
202
203 /// <summary>
204 /// <para>
205 /// Gets the link data part reference using the specified link index.
206 /// </para>
207 /// </summary>
208

```

```

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(ulong first, ulong second) => first == second;
260
261    /// <summary>
262    /// <para>
263    /// Determines whether this instance less than.
264    /// </para>
265    /// <para></para>
266    /// </summary>
267    /// <param name="first">
268    /// <para>The first.</para>
269    /// <para></para>
270    /// </param>
271    /// <param name="second">
272    /// <para>The second.</para>
273    /// <para></para>
274    /// </param>
275    /// <returns>
276    /// <para>The bool</para>
277    /// <para></para>
278    /// </returns>
279    [MethodImpl(MethodImplOptions.AggressiveInlining)]
280    protected override bool LessThan(ulong first, ulong second) => first < second;
281
282    /// <summary>
283    /// <para>
284    /// Determines whether this instance less or equal than.
285    /// </para>
286    /// <para></para>
287    /// </summary>

```

```

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

```

```

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

```

```

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

```

1.76 ./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        {

```

```

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     /// </param>
52     /// </returns>
53     /// <para>A ref raw link data part of t link</para>
54     /// <para></para>
55     /// </returns>
56     [MethodImpl(MethodImplOptions.AggressiveInlining)]
57     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
58         ↪ 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 }

```

1.77 ./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.Numbers;
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 avl balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizedAndThreadedAVLBalancedTreeMethods{TLink}" />
21     /// <seealso cref="ILinksTreeMethods{TLink}" />
22     public unsafe abstract class LinksAvlBalancedTreeMethodsBase<TLink> :
23         ↪ SizedAndThreadedAVLBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
26             ↪ UncheckedConverter<TLink, long>.Default;
27         private static readonly UncheckedConverter<TLink, int> _addressToInt32Converter =
28             ↪ UncheckedConverter<TLink, int>.Default;
29         private static readonly UncheckedConverter<bool, TLink> _boolToAddressConverter =
30             ↪ UncheckedConverter<bool, TLink>.Default;
31         private static readonly UncheckedConverter<TLink, bool> _addressToBoolConverter =
32             ↪ UncheckedConverter<TLink, bool>.Default;
33         private static readonly UncheckedConverter<int, TLink> _int32ToAddressConverter =
34             ↪ UncheckedConverter<int, TLink>.Default;
35
36         /// <summary>
37         /// <para>
38         /// The break.
39         /// </para>
40         /// <para></para>
41         /// </summary>

```



```

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

```

```

113 [MethodImpl(MethodImplOptions.AggressiveInlining)]
114 protected abstract TLink GetBasePartValue(TLink link);
115
116 /// <summary>
117 /// <para>
118 /// Determines whether this instance first is to the right of second.
119 /// </para>
120 /// <para></para>
121 /// </summary>
122 /// <param name="source">
123 /// <para>The source.</para>
124 /// <para></para>
125 /// </param>
126 /// <param name="target">
127 /// <para>The target.</para>
128 /// <para></para>
129 /// </param>
130 /// <param name="rootSource">
131 /// <para>The root source.</para>
132 /// <para></para>
133 /// </param>
134 /// <param name="rootTarget">
135 /// <para>The root target.</para>
136 /// <para></para>
137 /// </param>
138 /// <returns>
139 /// <para>The bool</para>
140 /// <para></para>
141 /// </returns>
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);
144
145 /// <summary>
146 /// <para>
147 /// Determines whether this instance first is to the left of second.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <param name="source">
152 /// <para>The source.</para>
153 /// <para></para>
154 /// </param>
155 /// <param name="target">
156 /// <para>The target.</para>
157 /// <para></para>
158 /// </param>
159 /// <param name="rootSource">
160 /// <para>The root source.</para>
161 /// <para></para>
162 /// </param>
163 /// <param name="rootTarget">
164 /// <para>The root target.</para>
165 /// <para></para>
166 /// </param>
167 /// <returns>
168 /// <para>The bool</para>
169 /// <para></para>
170 /// </returns>
171 [MethodImpl(MethodImplOptions.AggressiveInlining)]
172 protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);
173
174 /// <summary>
175 /// <para>
176 /// Gets the header reference.
177 /// </para>
178 /// <para></para>
179 /// </summary>
180 /// <returns>
181 /// <para>A ref links header of t link</para>
182 /// <para></para>
183 /// </returns>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLink>>(Header);
186
187 /// <summary>

```

```

188 /// <para>
189 /// Gets the link reference 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>A ref raw link of t link</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
    ↪ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));

203
204 /// <summary>
205 /// <para>
206 /// Gets the link values 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 list of t link</para>
216 /// <para></para>
217 /// </returns>
218 [MethodImpl(MethodImplOptions.AggressiveInlining)]
219 protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
220 {
221     ref var link = ref GetLinkReference(linkIndex);
222     return new Link<TLink>(linkIndex, link.Source, link.Target);
223 }
224
225 /// <summary>
226 /// <para>
227 /// Determines whether this instance first is to the left of second.
228 /// </para>
229 /// <para></para>
230 /// </summary>
231 /// <param name="first">
232 /// <para>The first.</para>
233 /// <para></para>
234 /// </param>
235 /// <param name="second">
236 /// <para>The second.</para>
237 /// <para></para>
238 /// </param>
239 /// <returns>
240 /// <para>The bool</para>
241 /// <para></para>
242 /// </returns>
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
245 {
246     ref var firstLink = ref GetLinkReference(first);
247     ref var secondLink = ref GetLinkReference(second);
248     return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
    ↪ secondLink.Source, secondLink.Target);
249 }
250
251 /// <summary>
252 /// <para>
253 /// Determines whether this instance first is to the right of second.
254 /// </para>
255 /// <para></para>
256 /// </summary>
257 /// <param name="first">
258 /// <para>The first.</para>
259 /// <para></para>
260 /// </param>
261 /// <param name="second">
262 /// <para>The second.</para>

```

```

263 /// <para></para>
264 /// </param>
265 /// <returns>
266 /// <para>The bool</para>
267 /// <para></para>
268 /// </returns>
269 [MethodImpl(MethodImplOptions.AggressiveInlining)]
270 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
271 {
272     ref var firstLink = ref GetLinkReference(first);
273     ref var secondLink = ref GetLinkReference(second);
274     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
275         ↪ 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,
294     ↪ -5);
295
296 /// <summary>
297 /// <para>
298 /// Sets the size value using the specified stored value.
299 /// </para>
300 /// <para></para>
301 /// </summary>
302 /// <param name="storedValue">
303 /// <para>The stored value.</para>
304 /// <para></para>
305 /// </param>
306 /// <param name="size">
307 /// <para>The size.</para>
308 /// <para></para>
309 /// </param>
310 [MethodImpl(MethodImplOptions.AggressiveInlining)]
311 protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
312     ↪ Bit<TLink>.PartialWrite(storedValue, size, 5, -5);
313
314 /// <summary>
315 /// <para>
316 /// Determines whether this instance get left is child value.
317 /// </para>
318 /// <para></para>
319 /// </summary>
320 /// <param name="value">
321 /// <para>The value.</para>
322 /// <para></para>
323 /// </param>
324 /// <returns>
325 /// <para>The bool</para>
326 /// <para></para>
327 /// </returns>
328 [MethodImpl(MethodImplOptions.AggressiveInlining)]
329 protected virtual bool GetLeftIsChildValue(TLink value)
330 {
331     unchecked
332     {
333         return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 4, 1));
334         //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 4, 1), default);
335     }
336 }
337
338 /// <summary>
339 /// <para>
340 /// Sets the left is child value using the specified stored value.

```

```

338    /// </para>
339    /// <para></para>
340    /// </summary>
341    /// <param name="storedValue">
342    /// <para>The stored value.</para>
343    /// <para></para>
344    /// </param>
345    /// <param name="value">
346    /// <para>The value.</para>
347    /// <para></para>
348    /// </param>
349    [MethodImpl(MethodImplOptions.AggressiveInlining)]
350    protected virtual void SetLeftIsChildValue(ref TLink storedValue, bool value)
351    {
352        unchecked
353        {
354            var previousValue = storedValue;
355            var modified = Bit<TLink>.PartialWrite(previousValue,
356                ↪ _boolToAddressConverter.Convert(value), 4, 1);
357            storedValue = modified;
358        }
359    }
360    /// <summary>
361    /// <para>
362    /// Determines whether this instance get right is child value.
363    /// </para>
364    /// <para></para>
365    /// </summary>
366    /// <param name="value">
367    /// <para>The value.</para>
368    /// <para></para>
369    /// </param>
370    /// <returns>
371    /// <para>The bool</para>
372    /// <para></para>
373    /// </returns>
374    [MethodImpl(MethodImplOptions.AggressiveInlining)]
375    protected virtual bool GetRightIsChildValue(TLink value)
376    {
377        unchecked
378        {
379            return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 3, 1));
380            //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 3, 1), default);
381        }
382    }
383    /// <summary>
384    /// <para>
385    /// Sets the right is child value using the specified stored value.
386    /// </para>
387    /// <para></para>
388    /// </summary>
389    /// <param name="storedValue">
390    /// <para>The stored value.</para>
391    /// <para></para>
392    /// </param>
393    /// <param name="value">
394    /// <para>The value.</para>
395    /// <para></para>
396    /// </param>
397    [MethodImpl(MethodImplOptions.AggressiveInlining)]
398    protected virtual void SetRightIsChildValue(ref TLink storedValue, bool value)
399    {
400        unchecked
401        {
402            var previousValue = storedValue;
403            var modified = Bit<TLink>.PartialWrite(previousValue,
404                ↪ _boolToAddressConverter.Convert(value), 3, 1);
405            storedValue = modified;
406        }
407    }
408    /// <summary>
409    /// <para>
410    /// Determines whether this instance is child.
411    /// </para>
412    /// <para></para>
413    /// </summary>

```

```

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

```

```

489 /// <para></para>
490 /// </summary>
491 public TLink this[TLink index]
492 {
493     [MethodImpl(MethodImplOptions.AggressiveInlining)]
494     get
495     {
496         var root = GetTreeRoot();
497         if (GreaterOrEqualThan(index, GetSize(root)))
498         {
499             return Zero;
500         }
501         while (!EqualToZero(root))
502         {
503             var left = GetLeftOrDefault(root);
504             var leftSize = GetSizeOrZero(left);
505             if (LessThan(index, leftSize))
506             {
507                 root = left;
508                 continue;
509             }
510             if (AreEqual(index, leftSize))
511             {
512                 return root;
513             }
514             root = GetRightOrDefault(root);
515             index = Subtract(index, Increment(leftSize));
516         }
517         return Zero; // TODO: Impossible situation exception (only if tree structure
                    ↳ broken)
518     }
519 }
520
521 /// <summary>
522 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
523 ↳ (концом).
524 /// </summary>
525 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
526 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
527 /// <returns>Индекс искомой связи.</returns>
528 [MethodImpl(MethodImplOptions.AggressiveInlining)]
529 public TLink Search(TLink source, TLink target)
530 {
531     var root = GetTreeRoot();
532     while (!EqualToZero(root))
533     {
534         ref var rootLink = ref GetLinkReference(root);
535         var rootSource = rootLink.Source;
536         var rootTarget = rootLink.Target;
537         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
538             ↳ node.Key < root.Key
539         {
540             root = GetLeftOrDefault(root);
541         }
542         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
543             ↳ node.Key > root.Key
544         {
545             root = GetRightOrDefault(root);
546         }
547         else // node.Key == root.Key
548         {
549             return root;
550         }
551     }
552     return Zero;
553 }
554
555 // TODO: Return indices range instead of references count
556 /// <summary>
557 /// <para>
558 /// Counts the usages using the specified link.
559 /// </para>
560 /// <para></para>
561 /// </summary>
562 /// <param name="link">
563 /// <para>The link.</para>
564 /// <para></para>
565 /// </param>

```

```

563 /// <returns>
564 /// <para>The link</para>
565 /// <para></para>
566 /// </returns>
567 [MethodImpl(MethodImplOptions.AggressiveInlining)]
568 public TLink CountUsages(TLink link)
569 {
570     var root = GetTreeRoot();
571     var total = GetSize(root);
572     var totalRightIgnore = Zero;
573     while (!EqualToZero(root))
574     {
575         var @base = GetBasePartValue(root);
576         if (LessOrEqualThan(@base, link))
577         {
578             root = GetRightOrDefault(root);
579         }
580         else
581         {
582             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
583             root = GetLeftOrDefault(root);
584         }
585     }
586     root = GetTreeRoot();
587     var totalLeftIgnore = Zero;
588     while (!EqualToZero(root))
589     {
590         var @base = GetBasePartValue(root);
591         if (GreaterOrEqualThan(@base, link))
592         {
593             root = GetLeftOrDefault(root);
594         }
595         else
596         {
597             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
598             root = GetRightOrDefault(root);
599         }
600     }
601     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
602 }
603
604
605 /// <summary>
606 /// <para>
607 /// Eaches the usage using the specified link.
608 /// </para>
609 /// <para></para>
610 /// </summary>
611 /// <param name="link">
612 /// <para>The link.</para>
613 /// <para></para>
614 /// </param>
615 /// <param name="handler">
616 /// <para>The handler.</para>
617 /// <para></para>
618 /// </param>
619 /// <returns>
620 /// <para>The continue.</para>
621 /// <para></para>
622 /// </returns>
623 [MethodImpl(MethodImplOptions.AggressiveInlining)]
624 public TLink EachUsage(TLink link, Func<IList<TLink>, TLink> handler)
625 {
626     var root = GetTreeRoot();
627     if (EqualToZero(root))
628     {
629         return Continue;
630     }
631     TLink first = Zero, current = root;
632     while (!EqualToZero(current))
633     {
634         var @base = GetBasePartValue(current);
635         if (GreaterOrEqualThan(@base, link))
636         {
637             if (AreEqual(@base, link))
638             {
639                 first = current;
640             }

```



```

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

```

1.78 ./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 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.United.Generic
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the links recursionless size balanced tree methods base.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}" />
20     /// <seealso cref="ILinksTreeMethods{TLink}" />
21     public unsafe abstract class LinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
22     ↪ RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
23     {
24         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
25         ↪ UncheckedConverter<TLink, long>.Default;
26     }
27 }

```

```

24
25     /// <summary>
26     /// <para>
27     /// The break.
28     /// </para>
29     /// <para></para>
30     /// </summary>
31     protected readonly TLink Break;
32     /// <summary>
33     /// <para>
34     /// The continue.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     protected readonly TLink Continue;
39     /// <summary>
40     /// <para>
41     /// The links.
42     /// </para>
43     /// <para></para>
44     /// </summary>
45     protected readonly byte* Links;
46     /// <summary>
47     /// <para>
48     /// The header.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     protected readonly byte* Header;
53
54     /// <summary>
55     /// <para>
56     /// Initializes a new <see cref="LinksRecursionlessSizeBalancedTreeMethodsBase"/>
57     ↪ instance.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="constants">
62     /// <para>A constants.</para>
63     /// <para></para>
64     /// </param>
65     /// <param name="links">
66     /// <para>A links.</para>
67     /// <para></para>
68     /// </param>
69     /// <param name="header">
70     /// <para>A header.</para>
71     /// <para></para>
72     /// </param>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
75     ↪ byte* links, byte* header)
76     {
77         Links = links;
78         Header = header;
79         Break = constants.Break;
80         Continue = constants.Continue;
81     }
82
83     /// <summary>
84     /// <para>
85     /// Gets the tree root.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <returns>
90     /// <para>The link</para>
91     /// <para></para>
92     /// </returns>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     protected abstract TLink GetTreeRoot();
95
96     /// <summary>
97     /// <para>
98     /// Gets the base part value using the specified link.
99     /// </para>
100    /// <para></para>
101    /// </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 GetBasePartValue(TLink link);
110
111     /// <summary>
112     /// <para>
113     /// Determines whether this instance first is to the right of second.
114     /// </para>
115     /// <para></para>
116     /// </summary>
117     /// <param name="source">
118     /// <para>The source.</para>
119     /// <para></para>
120     /// </param>
121     /// <param name="target">
122     /// <para>The target.</para>
123     /// <para></para>
124     /// </param>
125     /// <param name="rootSource">
126     /// <para>The root source.</para>
127     /// <para></para>
128     /// </param>
129     /// <param name="rootTarget">
130     /// <para>The root target.</para>
131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The bool</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
        ↪ rootSource, TLink rootTarget);
139
140     /// <summary>
141     /// <para>
142     /// Determines whether this instance first is to the left of second.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     /// <param name="source">
147     /// <para>The source.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="target">
151     /// <para>The target.</para>
152     /// <para></para>
153     /// </param>
154     /// <param name="rootSource">
155     /// <para>The root source.</para>
156     /// <para></para>
157     /// </param>
158     /// <param name="rootTarget">
159     /// <para>The root target.</para>
160     /// <para></para>
161     /// </param>
162     /// <returns>
163     /// <para>The bool</para>
164     /// <para></para>
165     /// </returns>
166     [MethodImpl(MethodImplOptions.AggressiveInlining)]
167     protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
        ↪ rootSource, TLink rootTarget);
168
169     /// <summary>
170     /// <para>
171     /// Gets the header reference.
172     /// </para>
173     /// <para></para>
174     /// </summary>
175     /// <returns>

```

```

176 /// <para>A ref links header of t link</para>
177 /// <para></para>
178 /// </returns>
179 [MethodImpl(MethodImplOptions.AggressiveInlining)]
180 protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↳ AsRef<LinksHeader<TLink>>(Header);

181
182 /// <summary>
183 /// <para>
184 /// Gets the link reference using the specified link.
185 /// </para>
186 /// <para></para>
187 /// </summary>
188 /// <param name="link">
189 /// <para>The link.</para>
190 /// <para></para>
191 /// </param>
192 /// <returns>
193 /// <para>A ref raw link of t link</para>
194 /// <para></para>
195 /// </returns>
196 [MethodImpl(MethodImplOptions.AggressiveInlining)]
197 protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
    ↳ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));

198
199 /// <summary>
200 /// <para>
201 /// Gets the link values using the specified link index.
202 /// </para>
203 /// <para></para>
204 /// </summary>
205 /// <param name="linkIndex">
206 /// <para>The link index.</para>
207 /// <para></para>
208 /// </param>
209 /// <returns>
210 /// <para>A list of t link</para>
211 /// <para></para>
212 /// </returns>
213 [MethodImpl(MethodImplOptions.AggressiveInlining)]
214 protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
215 {
216     ref var link = ref GetLinkReference(linkIndex);
217     return new Link<TLink>(linkIndex, link.Source, link.Target);
218 }
219
220 /// <summary>
221 /// <para>
222 /// Determines whether this instance first is to the left of second.
223 /// </para>
224 /// <para></para>
225 /// </summary>
226 /// <param name="first">
227 /// <para>The first.</para>
228 /// <para></para>
229 /// </param>
230 /// <param name="second">
231 /// <para>The second.</para>
232 /// <para></para>
233 /// </param>
234 /// <returns>
235 /// <para>The bool</para>
236 /// <para></para>
237 /// </returns>
238 [MethodImpl(MethodImplOptions.AggressiveInlining)]
239 protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
240 {
241     ref var firstLink = ref GetLinkReference(first);
242     ref var secondLink = ref GetLinkReference(second);
243     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↳ secondLink.Source, secondLink.Target);
244 }
245
246 /// <summary>
247 /// <para>
248 /// Determines whether this instance first is to the right of second.
249 /// </para>

```

```

250    /// <para></para>
251    /// </summary>
252    /// <param name="first">
253    /// <para>The first.</para>
254    /// <para></para>
255    /// </param>
256    /// <param name="second">
257    /// <para>The second.</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 first, TLink second)
266    {
267        ref var firstLink = ref GetLinkReference(first);
268        ref var secondLink = ref GetLinkReference(second);
269        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
270            ↪ secondLink.Source, secondLink.Target);
271    }
272    /// <summary>
273    /// <para>
274    /// The zero.
275    /// </para>
276    /// <para></para>
277    /// </summary>
278    public TLink this[TLink index]
279    {
280        [MethodImpl(MethodImplOptions.AggressiveInlining)]
281        get
282        {
283            var root = GetTreeRoot();
284            if (GreaterOrEqualThan(index, GetSize(root)))
285            {
286                return Zero;
287            }
288            while (!EqualToZero(root))
289            {
290                var left = GetLeftOrDefault(root);
291                var leftSize = GetSizeOrZero(left);
292                if (LessThan(index, leftSize))
293                {
294                    root = left;
295                    continue;
296                }
297                if (AreEqual(index, leftSize))
298                {
299                    return root;
300                }
301                root = GetRightOrDefault(root);
302                index = Subtract(index, Increment(leftSize));
303            }
304            return Zero; // TODO: Impossible situation exception (only if tree structure
305                ↪ broken)
306        }
307    }
308    /// <summary>
309    /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
310    ↪ (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)) //
325                ↪ node.Key < root.Key

```

```

324     {
325         root = GetLeftOrDefault(root);
326     }
327     else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
328         ↪ 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>

```

```

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

```

```

477     /// <summary>
478     /// <para>
479     /// Prints the node value using the specified node.
480     /// </para>
481     /// <para></para>
482     /// </summary>
483     /// <param name="node">
484     /// <para>The node.</para>
485     /// <para></para>
486     /// </param>
487     /// <param name="sb">
488     /// <para>The sb.</para>
489     /// <para></para>
490     /// </param>
491     [MethodImpl(MethodImplOptions.AggressiveInlining)]
492     protected override void PrintNodeValue(TLink node, StringBuilder sb)
493     {
494         ref var link = ref GetLinkReference(node);
495         sb.Append(' ');
496         sb.Append(link.Source);
497         sb.Append('-');
498         sb.Append('>');
499         sb.Append(link.Target);
500     }
501 }
502 }
```

```

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 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.United.Generic
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the links size balanced tree methods base.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="SizeBalancedTreeMethods{TLink}"/>
20     /// <seealso cref="ILinksTreeMethods{TLink}"/>
21     public unsafe abstract class LinksSizeBalancedTreeMethodsBase<TLink> :
22     ↪ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
23     {
24         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
25         ↪ UncheckedConverter<TLink, long>.Default;
26
27         /// <summary>
28         /// <para>
29         /// The break.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected readonly TLink Break;
34
35         /// <summary>
36         /// <para>
37         /// The continue.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         protected readonly TLink Continue;
42
43         /// <summary>
44         /// <para>
45         /// The links.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         protected readonly byte* Links;
50
51         /// <summary>
52         /// <para>
53         /// The header.
54         /// </para>
55         /// </summary>
56         protected readonly byte* Header;
57     }
58 }

```



```

50     /// <para></para>
51     /// </summary>
52     protected readonly byte* Header;
53
54     /// <summary>
55     /// <para>
56     /// Initializes a new <see cref="LinksSizeBalancedTreeMethodsBase"/> instance.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     /// <param name="constants">
61     /// <para>A constants.</para>
62     /// <para></para>
63     /// </param>
64     /// <param name="links">
65     /// <para>A links.</para>
66     /// <para></para>
67     /// </param>
68     /// <param name="header">
69     /// <para>A header.</para>
70     /// <para></para>
71     /// </param>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected LinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
74     ↪ byte* header)
75     {
76         Links = links;
77         Header = header;
78         Break = constants.Break;
79         Continue = constants.Continue;
80     }
81
82     /// <summary>
83     /// <para>
84     /// Gets the tree root.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <returns>
89     /// <para>The link</para>
90     /// <para></para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     protected abstract 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>

```

```

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

```

```

199    /// <summary>
200    /// <para>
201    /// Gets the link values using the specified link index.
202    /// </para>
203    /// <para></para>
204    /// </summary>
205    /// <param name="linkIndex">
206    /// <para>The link index.</para>
207    /// <para></para>
208    /// </param>
209    /// <returns>
210    /// <para>A list of t link</para>
211    /// <para></para>
212    /// </returns>
213    [MethodImpl(MethodImplOptions.AggressiveInlining)]
214    protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
215    {
216        ref var link = ref GetLinkReference(linkIndex);
217        return new Link<TLink>(linkIndex, link.Source, link.Target);
218    }
219
220    /// <summary>
221    /// <para>
222    /// Determines whether this instance first is to the left of second.
223    /// </para>
224    /// <para></para>
225    /// </summary>
226    /// <param name="first">
227    /// <para>The first.</para>
228    /// <para></para>
229    /// </param>
230    /// <param name="second">
231    /// <para>The second.</para>
232    /// <para></para>
233    /// </param>
234    /// <returns>
235    /// <para>The bool</para>
236    /// <para></para>
237    /// </returns>
238    [MethodImpl(MethodImplOptions.AggressiveInlining)]
239    protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
240    {
241        ref var firstLink = ref GetLinkReference(first);
242        ref var secondLink = ref GetLinkReference(second);
243        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
244            ↪ 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
274    /// <summary>
275    /// <para>
276    /// The zero.

```

```

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

```

```

349  /// </param>
350  /// <returns>
351  /// <para>The link</para>
352  /// <para></para>
353  /// </returns>
354  [MethodImpl(MethodImplOptions.AggressiveInlining)]
355  public TLink CountUsages(TLink link)
356  {
357      var root = GetTreeRoot();
358      var total = GetSize(root);
359      var totalRightIgnore = Zero;
360      while (!EqualToZero(root))
361      {
362          var @base = GetBasePartValue(root);
363          if (LessOrEqualThan(@base, link))
364          {
365              root = GetRightOrDefault(root);
366          }
367          else
368          {
369              totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
370              root = GetLeftOrDefault(root);
371          }
372      }
373      root = GetTreeRoot();
374      var totalLeftIgnore = Zero;
375      while (!EqualToZero(root))
376      {
377          var @base = GetBasePartValue(root);
378          if (GreaterOrEqualThan(@base, link))
379          {
380              root = GetLeftOrDefault(root);
381          }
382          else
383          {
384              totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
385              root = GetRightOrDefault(root);
386          }
387      }
388      return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
389  }
390
391  /// <summary>
392  /// <para>
393  /// Eaches the usage using the specified base.
394  /// </para>
395  /// <para></para>
396  /// </summary>
397  /// <param name="@base">
398  /// <para>The base.</para>
399  /// <para></para>
400  /// </param>
401  /// <param name="handler">
402  /// <para>The handler.</para>
403  /// <para></para>
404  /// </param>
405  /// <returns>
406  /// <para>The link</para>
407  /// <para></para>
408  /// </returns>
409  [MethodImpl(MethodImplOptions.AggressiveInlining)]
410  public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
411      ↪ EachUsageCore(@base, GetTreeRoot(), handler);
412
413  // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
414  ↪ low-level MSIL stack.
415  /// <summary>
416  /// <para>
417  /// Eaches the usage core using the specified base.
418  /// </para>
419  /// <para></para>
420  /// </summary>
421  /// <param name="@base">
422  /// <para>The base.</para>
423  /// <para></para>
424  /// </param>
425  /// <param name="link">
426  /// <para>The link.</para>

```

```

425 /// <para></para>
426 /// </param>
427 /// <param name="handler">
428 /// <para>The handler.</para>
429 /// <para></para>
430 /// </param>
431 /// <returns>
432 /// <para>The continue.</para>
433 /// <para></para>
434 /// </returns>
435 [MethodImpl(MethodImplOptions.AggressiveInlining)]
436 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
437 {
438     var @continue = Continue;
439     if (EqualToZero(link))
440     {
441         return @continue;
442     }
443     var linkBasePart = GetBasePartValue(link);
444     var @break = Break;
445     if (GreaterThan(linkBasePart, @base))
446     {
447         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
448         {
449             return @break;
450         }
451     }
452     else if (LessThan(linkBasePart, @base))
453     {
454         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
455         {
456             return @break;
457         }
458     }
459     else //if (linkBasePart == @base)
460     {
461         if (AreEqual(handler(GetLinkValues(link)), @break))
462         {
463             return @break;
464         }
465         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
466         {
467             return @break;
468         }
469         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
470         {
471             return @break;
472         }
473     }
474     return @continue;
475 }
476
477 /// <summary>
478 /// <para>
479 /// Prints the node value using the specified node.
480 /// </para>
481 /// <para></para>
482 /// </summary>
483 /// <param name="node">
484 /// <para>The node.</para>
485 /// <para></para>
486 /// </param>
487 /// <param name="sb">
488 /// <para>The sb.</para>
489 /// <para></para>
490 /// </param>
491 [MethodImpl(MethodImplOptions.AggressiveInlining)]
492 protected override void PrintNodeValue(TLink node, StringBuilder sb)
493 {
494     ref var link = ref GetLinkReference(node);
495     sb.Append(' ');
496     sb.Append(link.Source);
497     sb.Append(' - ');
498     sb.Append(' > ');
499     sb.Append(link.Target);
500 }
501 }
502 }

```

```

1.80 ./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>
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).LeftAsSource;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref link</para>
69         /// <para></para>
70         /// </returns>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         protected override ref TLink GetRightReference(TLink node) => ref
73             ↳ GetLinkReference(node).RightAsSource;
74
75         /// <summary>
76         /// <para>
77         /// Gets the left using the specified node.
78         /// </para>
79         /// </summary>
80         /// <param name="node">
81         /// <para>The node.</para>
82         /// <para></para>
83         /// </param>
84         /// <returns>
85         /// <para>The ref link</para>
86         /// <para></para>
87         /// </returns>
88         [MethodImpl(MethodImplOptions.AggressiveInlining)]
89         protected override ref TLink GetLeftUsing(TLink node) => ref
90             ↳ GetLinkReference(node).LeftAsSource;
91
92         /// <summary>
93         /// <para>
94         /// Gets the right using the specified node.
95         /// </para>
96         /// </summary>
97         /// <param name="node">
98         /// <para>The node.</para>
99         /// <para></para>
100        /// </param>
101        /// <returns>
102        /// <para>The ref link</para>
103        /// <para></para>
104        /// </returns>
105        [MethodImpl(MethodImplOptions.AggressiveInlining)]
106        protected override ref TLink GetRightUsing(TLink node) => ref
107            ↳ GetLinkReference(node).RightAsSource;
108    }
109 }

```

```

74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="node">
78     /// <para>The node.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override 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>

```



```

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

```

```

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

```

```

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 FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
333     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
334     ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
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(firstSource, secondSource) ||
365     ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
366
367     /// <summary>
368     /// <para>
369     /// Clears the node using the specified node.
370     /// </para>
371     /// <para></para>
372     /// </summary>
373     /// <param name="node">
374     /// <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 }

```

1.81 ./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> :
15        ↳ LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="LinksSourcesRecursionlessSizeBalancedTreeMethods"/>
20        ↳ instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="links">
29        /// <para>A links.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="header">
33        /// <para>A header.</para>
34        /// <para></para>
35        /// </param>
36        [MethodImpl(MethodImplOptions.AggressiveInlining)]
37        public LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<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).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">

```

```

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     /// </summary>
145     /// <param name="node">
146     /// <para>The node.</para>
147     /// </para>
148     /// </param>
149     /// <returns>
150     /// <para>The link</para>
151     /// </para>
152     /// </returns>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsSource;
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>
165     /// </param>
166     /// <param name="size">
167     /// <para>The size.</para>
168     /// </para>
169     /// </param>
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     protected override void SetSize(TLink node, TLink size) =>
172         ↪ GetLinkReference(node).SizeAsSource = size;
173
174     /// <summary>
175     /// <para>
176     /// Gets the tree root.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <returns>
181     /// <para>The link</para>
182     /// </para>
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     /// <para></para>
192     /// </summary>
193     /// <param name="link">
194     /// <para>The link.</para>
195     /// </para>
196     /// </param>
197     /// <returns>
198     /// <para>The link</para>
199     /// </para>
200     /// </returns>
201     [MethodImpl(MethodImplOptions.AggressiveInlining)]
202     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
203
204     /// <summary>
205     /// <para>
206     /// Determines whether this instance first is to the left of second.
207     /// </para>
208     /// <para></para>
209     /// </summary>
210     /// <param name="firstSource">
211     /// <para>The first source.</para>
212     /// </para>
213     /// </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,
    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
    ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));

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

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

```

1.82 ./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> :
15        ↳ LinksSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="LinksSourcesSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="header">
32        /// <para>A header.</para>
33        /// <para></para>
34        /// </param>
35        [MethodImpl(MethodImplOptions.AggressiveInlining)]
36        public LinksSourcesSizeBalancedTreeMethods(LinksConstants<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).LeftAsSource;
56
57        /// <summary>
58        /// <para>
59        /// Gets the right reference using the specified node.
60        /// </para>
61        /// <para></para>
62        /// </summary>
63        /// <param name="node">
64        /// <para>The node.</para>
65        /// <para></para>
66        /// </param>
67        /// <returns>
68        /// <para>The ref link</para>
69        /// <para></para>
70        /// </returns>
71        [MethodImpl(MethodImplOptions.AggressiveInlining)]
72        protected override ref TLink GetRightReference(TLink node) => ref
73            ↳ GetLinkReference(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 left link</para>
87        /// <para></para>
88        /// </returns>
89        [MethodImpl(MethodImplOptions.AggressiveInlining)]
90        protected override ref TLink GetLeftLink(TLink node) => ref
91            ↳ GetLinkReference(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 right link</para>
105       /// <para></para>
106       /// </returns>
107       [MethodImpl(MethodImplOptions.AggressiveInlining)]
108       protected override ref TLink GetRightLink(TLink node) => ref
109           ↳ GetLinkReference(node).RightAsSource;
110    }
111 }

```



```

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;

```

```

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(TLink node, TLink size) =>
171     ↪ GetLinkReference(node).SizeAsSource = size;
172
173     /// <summary>
174     /// <para>
175     /// Gets the tree root.
176     /// </para>
177     /// <para></para>
178     /// </summary>
179     /// <returns>
180     /// <para>The link</para>
181     /// <para></para>
182     /// </returns>
183     [MethodImpl(MethodImplOptions.AggressiveInlining)]
184     protected override 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 FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
    ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
231
232     /// <summary>
233     /// <para>
234     /// Determines whether this instance first is to the right of second.
235     /// </para>
236     /// <para></para>
237     /// </summary>
238     /// <param name="firstSource">
239     /// <para>The first source.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="firstTarget">
243     /// <para>The first target.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="secondSource">
247     /// <para>The second source.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondTarget">
251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(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 }

```

1.83 ./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> :
    ↪ LinksAvlBalancedTreeMethodsBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="LinksTargetsAvlBalancedTreeMethods" /> instance.
19         /// </para>
20         /// <para></para>

```

```

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

```

```

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>
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) =>
157        ↪ GetSizeValue(GetLinkReference(node).SizeAsTarget);
158
159    /// <summary>
160    /// <para>
161    /// Sets the size using the specified node.
162    /// </para>
163    /// <para></para>
164    /// </summary>
165    /// <param name="node">
166    /// <para>The node.</para>
167    /// <para></para>
168    /// </param>
169    /// <param name="size">
170    /// <para>The size.</para>
171    /// <para></para>
172    /// </param>
173    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

171 protected override void SetSize(TLink node, TLink size) => SetSizeValue(ref
    ↳ GetLinkReference(node).SizeAsTarget, size);
172
173 /// <summary>
174 /// <para>
175 /// Determines whether this instance get left is child.
176 /// </para>
177 /// <para></para>
178 /// </summary>
179 /// <param name="node">
180 /// <para>The node.</para>
181 /// <para></para>
182 /// </param>
183 /// <returns>
184 /// <para>The bool</para>
185 /// <para></para>
186 /// </returns>
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 protected override bool GetLeftIsChild(TLink node) =>
    ↳ GetLeftIsChildValue(GetLinkReference(node).SizeAsTarget);
189
190 /// <summary>
191 /// <para>
192 /// Sets the left is child using the specified node.
193 /// </para>
194 /// <para></para>
195 /// </summary>
196 /// <param name="node">
197 /// <para>The node.</para>
198 /// <para></para>
199 /// </param>
200 /// <param name="value">
201 /// <para>The value.</para>
202 /// <para></para>
203 /// </param>
204 [MethodImpl(MethodImplOptions.AggressiveInlining)]
205 protected override void SetLeftIsChild(TLink node, bool value) =>
    ↳ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
206
207 /// <summary>
208 /// <para>
209 /// Determines whether this instance get right is child.
210 /// </para>
211 /// <para></para>
212 /// </summary>
213 /// <param name="node">
214 /// <para>The node.</para>
215 /// <para></para>
216 /// </param>
217 /// <returns>
218 /// <para>The bool</para>
219 /// <para></para>
220 /// </returns>
221 [MethodImpl(MethodImplOptions.AggressiveInlining)]
222 protected override bool GetRightIsChild(TLink node) =>
    ↳ GetRightIsChildValue(GetLinkReference(node).SizeAsTarget);
223
224 /// <summary>
225 /// <para>
226 /// Sets the right is child using the specified node.
227 /// </para>
228 /// <para></para>
229 /// </summary>
230 /// <param name="node">
231 /// <para>The node.</para>
232 /// <para></para>
233 /// </param>
234 /// <param name="value">
235 /// <para>The value.</para>
236 /// <para></para>
237 /// </param>
238 [MethodImpl(MethodImplOptions.AggressiveInlining)]
239 protected override void SetRightIsChild(TLink node, bool value) =>
    ↳ SetRightIsChildValue(ref GetLinkReference(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(TLink node) =>
257         ↳ GetBalanceValue(GetLinkReference(node).SizeAsTarget);
258
259     /// <summary>
260     /// <para>
261     /// Sets the balance using the specified node.
262     /// </para>
263     /// <para></para>
264     /// </summary>
265     /// <param name="node">
266     /// <para>The node.</para>
267     /// <para></para>
268     /// </param>
269     /// <param name="value">
270     /// <para>The value.</para>
271     /// <para></para>
272     /// </param>
273     [MethodImpl(MethodImplOptions.AggressiveInlining)]
274     protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
275         ↳ GetLinkReference(node).SizeAsTarget, value);
276
277     /// <summary>
278     /// <para>
279     /// Gets the tree root.
280     /// </para>
281     /// <para></para>
282     /// </summary>
283     /// <returns>
284     /// <para>The link</para>
285     /// <para></para>
286     /// </returns>
287     [MethodImpl(MethodImplOptions.AggressiveInlining)]
288     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
289
290     /// <summary>
291     /// <para>
292     /// Gets the base part value using the specified link.
293     /// </para>
294     /// <para></para>
295     /// </summary>
296     /// <param name="link">
297     /// <para>The link.</para>
298     /// <para></para>
299     /// </param>
300     /// <returns>
301     /// <para>The link</para>
302     /// <para></para>
303     /// </returns>
304     [MethodImpl(MethodImplOptions.AggressiveInlining)]
305     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Target;
306
307     /// <summary>
308     /// <para>
309     /// Determines whether this instance first is to the left of second.
310     /// </para>
311     /// <para></para>
312     /// </summary>
313     /// <param name="firstSource">
314     /// <para>The first source.</para>
315     /// <para></para>
316     /// </param>
317     /// <param name="firstTarget">
318     /// <para>The first target.</para>
319     /// <para></para>
320     /// </param>

```

```

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(firstTarget, secondTarget) ||
    ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
333
334     /// <summary>
335     /// <para>
336     /// Determines whether this instance first is to the right of second.
337     /// </para>
338     /// <para></para>
339     /// </summary>
340     /// <param name="firstSource">
341     /// <para>The first source.</para>
342     /// <para></para>
343     /// </param>
344     /// <param name="firstTarget">
345     /// <para>The first target.</para>
346     /// <para></para>
347     /// </param>
348     /// <param name="secondSource">
349     /// <para>The second source.</para>
350     /// <para></para>
351     /// </param>
352     /// <param name="secondTarget">
353     /// <para>The second target.</para>
354     /// <para></para>
355     /// </param>
356     /// <returns>
357     /// <para>The bool</para>
358     /// <para></para>
359     /// </returns>
360     [MethodImpl(MethodImplOptions.AggressiveInlining)]
361     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
    ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
362
363     /// <summary>
364     /// <para>
365     /// Clears the node using the specified node.
366     /// </para>
367     /// <para></para>
368     /// </summary>
369     /// <param name="node">
370     /// <para>The node.</para>
371     /// <para></para>
372     /// </param>
373     [MethodImpl(MethodImplOptions.AggressiveInlining)]
374     protected override void ClearNode(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 }

```

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

```

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         /// <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
74         ↪ GetLinkReference(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>

```

```

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>
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).SizeAsTarget;
157    /// <summary>
158    /// <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>
226     /// <returns>
227     /// <para>The bool</para>
228     /// <para></para>
229     /// </returns>
230     [MethodImpl(MethodImplOptions.AggressiveInlining)]
231     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
232         ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
        ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));

```

```

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

```

1.85 ./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> :
        ↪ LinksSizeBalancedTreeMethodsBase<TLink>
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="LinksTargetsSizeBalancedTreeMethods" /> instance.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <param name="constants">
23        /// <para>A constants.</para>
24        /// <para></para>
25        /// </param>

```

```

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

```

```

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>
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).SizeAsTarget;
157
158     /// <summary>
159     /// <para>
160     /// Sets the size using the specified node.
161     /// </para>
162     /// <para></para>
163     /// </summary>
164     /// <param name="node">
165     /// <para>The node.</para>
166     /// <para></para>
167     /// </param>
168     /// <param name="size">
169     /// <para>The size.</para>
170     /// <para></para>
171     /// </param>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     protected override void SetSize(TLink node, TLink size) =>
174         ↪ GetLinkReference(node).SizeAsTarget = size;
175
176     /// <summary>
177     /// <para>
178     /// Gets the tree root.

```

```

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

```

```

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 }

```

1.86 ./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>
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
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)]

```



```

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
            ↳ LinksSourcesAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
104         _createTargetTreeMethods = () => new
            ↳ LinksTargetsAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
105     }
106     else
107     {
108         _createSourceTreeMethods = () => new
            ↳ LinksSourcesSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
109         _createTargetTreeMethods = () => new
            ↳ LinksTargetsSizeBalancedTreeMethods<TLink>(Constants, _links, _header);

```

```

110     }
111     Init(memory, memoryReservationStep);
112 }
113
114 /// <summary>
115 /// <para>
116 /// Sets the pointers using the specified memory.
117 /// </para>
118 /// <para></para>
119 /// </summary>
120 /// <param name="memory">
121 /// <para>The memory.</para>
122 /// <para></para>
123 /// </param>
124 [MethodImpl(MethodImplOptions.AggressiveInlining)]
125 protected override void SetPointers(IResizableDirectMemory memory)
126 {
127     _links = (byte*)memory.Pointer;
128     _header = _links;
129     SourcesTreeMethods = _createSourceTreeMethods();
130     TargetsTreeMethods = _createTargetTreeMethods();
131     UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_links, _header);
132 }
133
134 /// <summary>
135 /// <para>
136 /// Resets the pointers.
137 /// </para>
138 /// <para></para>
139 /// </summary>
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 protected override void ResetPointers()
142 {
143     base.ResetPointers();
144     _links = null;
145     _header = null;
146 }
147
148 /// <summary>
149 /// <para>
150 /// Gets the header reference.
151 /// </para>
152 /// <para></para>
153 /// </summary>
154 /// <returns>
155 /// <para>A ref links header of t link</para>
156 /// <para></para>
157 /// </returns>
158 [MethodImpl(MethodImplOptions.AggressiveInlining)]
159 protected override ref LinksHeader<TLink> GetHeaderReference() => ref
160     ↪ AsRef<LinksHeader<TLink>>(_header);
161
162 /// <summary>
163 /// <para>
164 /// Gets the link reference using the specified link index.
165 /// </para>
166 /// <para></para>
167 /// </summary>
168 /// <param name="linkIndex">
169 /// <para>The link index.</para>
170 /// <para></para>
171 /// </param>
172 /// <returns>
173 /// <para>A ref raw link of t link</para>
174 /// <para></para>
175 /// </returns>
176 [MethodImpl(MethodImplOptions.AggressiveInlining)]
177 protected override ref RawLink<TLink> GetLinkReference(TLink linkIndex) => ref
178     ↪ AsRef<RawLink<TLink>>(_links + (LinkSizeInBytes * ConvertToInt64(linkIndex)));

```

1.87 ./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
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 united memory links base.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     /// <seealso cref="DisposableBase"/>
22     /// <seealso cref="ILinks{TLink}"/>
23     public abstract class UnitedMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink>
24     {
25         private static readonly EqualityComparer<TLink> _equalityComparer =
26             EqualityComparer<TLink>.Default;
27         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
28         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
29             UncheckedConverter<TLink, long>.Default;
30         private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
31             UncheckedConverter<long, TLink>.Default;
32
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         /// <para></para>
48         /// </summary>
49         public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
50
51         /// <summary>
52         /// <para>
53         /// The link size in bytes.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
58
59         /// <summary>
60         /// <para>
61         /// The memory.
62         /// </para>
63         /// <para></para>
64         /// </summary>
65         protected readonly IResizableDirectMemory _memory;
66
67         /// <summary>
68         /// <para>
69         /// The memory reservation step.
70         /// </para>
71         /// <para></para>
72         /// </summary>
73         protected readonly long _memoryReservationStep;
74
75         /// <summary>
76         /// <para>
77         /// The targets tree methods.
78         /// </para>
79         /// <para></para>
80         /// </summary>
81         protected ILinksTreeMethods<TLink> TargetsTreeMethods;
82
83         /// <summary>
84         /// <para>
85         /// The sources tree methods.
86         /// </para>
87         /// <para></para>
88         /// </summary>

```

```

84     protected ILinksTreeMethods<TLink> SourcesTreeMethods;
85     // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
    ↪     нужно использовать не список а дерево, так как так можно быстрее проверить на
    ↪     наличие связи внутри
86     /// <summary>
87     /// <para>
88     /// The unused links list methods.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     protected ILinksListMethods<TLink> UnusedLinksListMethods;
93
94     /// <summary>
95     /// Возвращает общее число связей находящихся в хранилище.
96     /// </summary>
97     protected virtual TLink Total
98     {
99         [MethodImpl(MethodImplOptions.AggressiveInlining)]
100         get
101         {
102             ref var header = ref GetHeaderReference();
103             return Subtract(header.AllocatedLinks, header.FreeLinks);
104         }
105     }
106
107     /// <summary>
108     /// <para>
109     /// Gets the constants value.
110     /// </para>
111     /// <para></para>
112     /// </summary>
113     public virtual LinksConstants<TLink> Constants
114     {
115         [MethodImpl(MethodImplOptions.AggressiveInlining)]
116         get;
117     }
118
119     /// <summary>
120     /// <para>
121     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
122     /// </para>
123     /// <para></para>
124     /// </summary>
125     /// <param name="memory">
126     /// <para>A memory.</para>
127     /// <para></para>
128     /// </param>
129     /// <param name="memoryReservationStep">
130     /// <para>A memory reservation step.</para>
131     /// <para></para>
132     /// </param>
133     /// <param name="constants">
134     /// <para>A constants.</para>
135     /// <para></para>
136     /// </param>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
    ↪     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 restrictions.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <param name="restrictions">
198 /// <para>The restrictions.</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(ICollection<TLink> restrictions)
211 {
212     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
213     if (restrictions.Count == 0)
214     {
215         return Total;
216     }
217     var constants = Constants;
218     var any = constants.Any;
219     var index = restrictions[constants.IndexPart];
220     if (restrictions.Count == 1)
221     {
222         if (AreEqual(index, any))
223         {
224             return Total;
225         }
226         return Exists(index) ? GetOne() : GetZero();
227     }
228     if (restrictions.Count == 2)
229     {
230         var value = restrictions[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 if (restrictions.Count == 3)
259 {
260     var source = restrictions[constants.SourcePart];
261     var target = restrictions[constants.TargetPart];
262     if (AreEqual(index, any))
263     {
264         if (AreEqual(source, any) && AreEqual(target, any))
265         {
266             return Total;
267         }
268         else if (AreEqual(source, any))
269         {
270             return TargetsTreeMethods.CountUsages(target);
271         }
272         else if (AreEqual(target, any))
273         {
274             return SourcesTreeMethods.CountUsages(source);
275         }
276         else //if(source != Any && target != Any)
277         {
278             // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
279             var link = SourcesTreeMethods.Search(source, target);
280             return AreEqual(link, constants.Null) ? GetZero() : GetOne();
281         }
282     }
283     else
284     {
285         if (!Exists(index))
286         {
287             return GetZero();
288         }
289         if (AreEqual(source, any) && AreEqual(target, any))
290         {
291             return GetOne();
292         }
293         ref var storedLinkValue = ref GetLinkReference(index);
294         if (!AreEqual(source, any) && !AreEqual(target, any))
295         {
296             if (AreEqual(storedLinkValue.Source, source) &&
297                 ↪ AreEqual(storedLinkValue.Target, target))
298             {
299                 return GetOne();
300             }
301             return GetZero();
302         }
303         var value = default(TLink);
304         if (AreEqual(source, any))
305         {
306             value = target;
307         }
308         if (AreEqual(target, any))
309         {
310             value = source;

```

```

309     }
310     if (AreEqual(storedLinkValue.Source, value) ||
        ↪ AreEqual(storedLinkValue.Target, value))
311     {
312         return GetOne();
313     }
314     return GetZero();
315 }
316 }
317 throw new NotSupportedException("Другие размеры и способы ограничений не
        ↪ поддерживаются.");
318 }
319
320 /// <summary>
321 /// <para>
322 /// Eaches the handler.
323 /// </para>
324 /// <para></para>
325 /// </summary>
326 /// <param name="handler">
327 /// <para>The handler.</para>
328 /// <para></para>
329 /// </param>
330 /// <param name="restrictions">
331 /// <para>The restrictions.</para>
332 /// <para></para>
333 /// </param>
334 /// <exception cref="NotSupportedException">
335 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
336 /// <para></para>
337 /// </exception>
338 /// <returns>
339 /// <para>The link</para>
340 /// <para></para>
341 /// </returns>
342 [MethodImpl(MethodImplOptions.AggressiveInlining)]
343 public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
344 {
345     var constants = Constants;
346     var @break = constants.Break;
347     if (restrictions.Count == 0)
348     {
349         for (var link = GetOne(); LessOrEqualThan(link,
            ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
350         {
351             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
352             {
353                 return @break;
354             }
355         }
356         return @break;
357     }
358     var @continue = constants.Continue;
359     var any = constants.Any;
360     var index = restrictions[constants.IndexPart];
361     if (restrictions.Count == 1)
362     {
363         if (AreEqual(index, any))
364         {
365             return Each(handler, Array.Empty<TLink>());
366         }
367         if (!Exists(index))
368         {
369             return @continue;
370         }
371         return handler(GetLinkStruct(index));
372     }
373     if (restrictions.Count == 2)
374     {
375         var value = restrictions[1];
376         if (AreEqual(index, any))
377         {
378             if (AreEqual(value, any))
379             {
380                 return Each(handler, Array.Empty<TLink>());
381             }
382             if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
383             {

```

```

384         return @break;
385     }
386     return Each(handler, new Link<TLink>(index, any, value));
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 (restrictions.Count == 3)
408 {
409     var source = restrictions[constants.SourcePart];
410     var target = restrictions[constants.TargetPart];
411     if (AreEqual(index, any))
412     {
413         if (AreEqual(source, any) && AreEqual(target, any))
414         {
415             return Each(handler, Array.Empty<TLink>());
416         }
417         else if (AreEqual(source, any))
418         {
419             return TargetsTreeMethods.EachUsage(target, handler);
420         }
421         else if (AreEqual(target, any))
422         {
423             return SourcesTreeMethods.EachUsage(source, handler);
424         }
425         else //if(source != Any && target != Any)
426         {
427             var link = SourcesTreeMethods.Search(source, target);
428             return AreEqual(link, constants.Null) ? @continue :
429                 ↪ handler(GetLinkStruct(link));
430         }
431     }
432     else
433     {
434         if (!Exists(index))
435         {
436             return @continue;
437         }
438         if (AreEqual(source, any) && AreEqual(target, any))
439         {
440             return handler(GetLinkStruct(index));
441         }
442         ref var storedLinkValue = ref GetLinkReference(index);
443         if (!AreEqual(source, any) && !AreEqual(target, any))
444         {
445             if (AreEqual(storedLinkValue.Source, source) &&
446                 AreEqual(storedLinkValue.Target, target))
447             {
448                 return handler(GetLinkStruct(index));
449             }
450             return @continue;
451         }
452         var value = default(TLink);
453         if (AreEqual(source, any))
454         {
455             value = target;
456         }
457         if (AreEqual(target, any))
458         {
459             value = source;
460         }
461         if (AreEqual(storedLinkValue.Source, value) ||

```



```

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
    ↳ в другом месте (но не в менеджере памяти, а в логике Links)
473 /// </remarks>
474 [MethodImpl(MethodImplOptions.AggressiveInlining)]
475 public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
476 {
477     var constants = Constants;
478     var @null = constants.Null;
479     var linkIndex = restrictions[constants.IndexPart];
480     ref var link = ref GetLinkReference(linkIndex);
481     ref var header = ref GetHeaderReference();
482     ref var firstAsSource = ref header.RootAsSource;
483     ref var firstAsTarget = ref header.RootAsTarget;
484     // Будет корректно работать только в том случае, если пространство выделенной связи
    ↳ предварительно заполнено нулями
485     if (!AreEqual(link.Source, @null))
486     {
487         SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
488     }
489     if (!AreEqual(link.Target, @null))
490     {
491         TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
492     }
493     link.Source = substitution[constants.SourcePart];
494     link.Target = substitution[constants.TargetPart];
495     if (!AreEqual(link.Source, @null))
496     {
497         SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
498     }
499     if (!AreEqual(link.Target, @null))
500     {
501         TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
502     }
503     return linkIndex;
504 }
505
506 /// <remarks>
507 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
    ↳ пространство
508 /// </remarks>
509 [MethodImpl(MethodImplOptions.AggressiveInlining)]
510 public virtual TLink Create(IList<TLink> restrictions)
511 {
512     ref var header = ref GetHeaderReference();
513     var freeLink = header.FirstFreeLink;
514     if (!AreEqual(freeLink, Constants.Null))
515     {
516         UnusedLinksListMethods.Detach(freeLink);
517     }
518     else
519     {
520         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
521         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
522         {
523             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
524         }
525         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
526         {
527             _memory.ReservedCapacity += _memory.ReservationStep;
528             SetPointers(_memory);
529             header = ref GetHeaderReference();
530             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
    ↳ LinkSizeInBytes);
531         }
532         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
533         _memory.UsedCapacity += LinkSizeInBytes;
534     }

```

```

535     return freeLink;
536 }
537
538 /// <summary>
539 /// <para>
540 /// Deletes the restrictions.
541 /// </para>
542 /// <para></para>
543 /// </summary>
544 /// <param name="restrictions">
545 /// <para>The restrictions.</para>
546 /// <para></para>
547 /// </param>
548 [MethodImpl(MethodImplOptions.AggressiveInlining)]
549 public virtual void Delete(ICollection<TLink> restrictions)
550 {
551     ref var header = ref GetHeaderReference();
552     var link = restrictions[Constants.IndexPart];
553     if (LessThan(link, header.AllocatedLinks))
554     {
555         UnusedLinksListMethods.AttachAsFirst(link);
556     }
557     else if (AreEqual(link, header.AllocatedLinks))
558     {
559         header.AllocatedLinks = Decrement(header.AllocatedLinks);
560         _memory.UsedCapacity -= LinkSizeInBytes;
561         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
562         // ↳ пока не дойдём до первой существующей связи
563         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
564         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
565             ↳ IsUnusedLink(header.AllocatedLinks))
566         {
567             UnusedLinksListMethods.Detach(header.AllocatedLinks);
568             header.AllocatedLinks = Decrement(header.AllocatedLinks);
569             _memory.UsedCapacity -= LinkSizeInBytes;
570         }
571     }
572 }
573
574 /// <summary>
575 /// <para>
576 /// Gets the link struct using the specified link index.
577 /// </para>
578 /// <para></para>
579 /// </summary>
580 /// <param name="linkIndex">
581 /// <para>The link index.</para>
582 /// <para></para>
583 /// </param>
584 /// <returns>
585 /// <para>A list of t link</para>
586 /// <para></para>
587 /// </returns>
588 [MethodImpl(MethodImplOptions.AggressiveInlining)]
589 public ICollection<TLink> GetLinkStruct(TLink linkIndex)
590 {
591     ref var link = ref GetLinkReference(linkIndex);
592     return new Link<TLink>(linkIndex, link.Source, link.Target);
593 }
594
595 /// <remarks>
596 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
597 /// ↳ адрес реально поменялся
598 ///
599 /// Указатель this.links может быть в том же месте,
600 /// так как 0-я связь не используется и имеет такой же размер как Header,
601 /// поэтому header размещается в том же месте, что и 0-я связь
602 /// </remarks>
603 [MethodImpl(MethodImplOptions.AggressiveInlining)]
604 protected abstract void SetPointers(IResizableDirectMemory memory);
605
606 /// <summary>
607 /// <para>
608 /// Resets the pointers.
609 /// </para>
610 /// <para></para>
611 /// </summary>
612 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

610 protected virtual void ResetPointers()
611 {
612     SourcesTreeMethods = null;
613     TargetsTreeMethods = null;
614     UnusedLinksListMethods = null;
615 }
616
617 /// <summary>
618 /// <para>
619 /// Gets the header reference.
620 /// </para>
621 /// <para></para>
622 /// </summary>
623 /// <returns>
624 /// <para>A ref links header of t link</para>
625 /// <para></para>
626 /// </returns>
627 [MethodImpl(MethodImplOptions.AggressiveInlining)]
628 protected abstract ref LinksHeader<TLink> GetHeaderReference();
629
630 /// <summary>
631 /// <para>
632 /// Gets the link reference using the specified link index.
633 /// </para>
634 /// <para></para>
635 /// </summary>
636 /// <param name="linkIndex">
637 /// <para>The link index.</para>
638 /// <para></para>
639 /// </param>
640 /// <returns>
641 /// <para>A ref raw link of t link</para>
642 /// <para></para>
643 /// </returns>
644 [MethodImpl(MethodImplOptions.AggressiveInlining)]
645 protected abstract ref RawLink<TLink> GetLinkReference(TLink linkIndex);
646
647 /// <summary>
648 /// <para>
649 /// Determines whether this instance exists.
650 /// </para>
651 /// <para></para>
652 /// </summary>
653 /// <param name="link">
654 /// <para>The link.</para>
655 /// <para></para>
656 /// </param>
657 /// <returns>
658 /// <para>The bool</para>
659 /// <para></para>
660 /// </returns>
661 [MethodImpl(MethodImplOptions.AggressiveInlining)]
662 protected virtual bool Exists(TLink link)
663     => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
664     && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
665     && !IsUnusedLink(link);
666
667 /// <summary>
668 /// <para>
669 /// Determines whether this instance is unused link.
670 /// </para>
671 /// <para></para>
672 /// </summary>
673 /// <param name="linkIndex">
674 /// <para>The link index.</para>
675 /// <para></para>
676 /// </param>
677 /// <returns>
678 /// <para>The bool</para>
679 /// <para></para>
680 /// </returns>
681 [MethodImpl(MethodImplOptions.AggressiveInlining)]
682 protected virtual bool IsUnusedLink(TLink linkIndex)
683 {
684     if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
685         ↪ is not needed
686     {
687         ref var link = ref GetLinkReference(linkIndex);

```

```

687         return AreEqual(link.SizeAsSource, default) && !AreEqual(link.Source, default);
688     }
689     else
690     {
691         return true;
692     }
693 }
694
695 /// <summary>
696 /// <para>
697 /// Gets the one.
698 /// </para>
699 /// <para></para>
700 /// </summary>
701 /// <returns>
702 /// <para>The link</para>
703 /// <para></para>
704 /// </returns>
705 [MethodImpl(MethodImplOptions.AggressiveInlining)]
706 protected virtual TLink GetOne() => _one;
707
708 /// <summary>
709 /// <para>
710 /// Gets the zero.
711 /// </para>
712 /// <para></para>
713 /// </summary>
714 /// <returns>
715 /// <para>The link</para>
716 /// <para></para>
717 /// </returns>
718 [MethodImpl(MethodImplOptions.AggressiveInlining)]
719 protected virtual TLink GetZero() => default;
720
721 /// <summary>
722 /// <para>
723 /// Determines whether this instance are equal.
724 /// </para>
725 /// <para></para>
726 /// </summary>
727 /// <param name="first">
728 /// <para>The first.</para>
729 /// <para></para>
730 /// </param>
731 /// <param name="second">
732 /// <para>The second.</para>
733 /// <para></para>
734 /// </param>
735 /// <returns>
736 /// <para>The bool</para>
737 /// <para></para>
738 /// </returns>
739 [MethodImpl(MethodImplOptions.AggressiveInlining)]
740 protected virtual bool AreEqual(TLink first, TLink second) =>
741     ↪ _equalityComparer.Equals(first, second);
742
743 /// <summary>
744 /// <para>
745 /// Determines whether this instance less than.
746 /// </para>
747 /// <para></para>
748 /// </summary>
749 /// <param name="first">
750 /// <para>The first.</para>
751 /// <para></para>
752 /// </param>
753 /// <param name="second">
754 /// <para>The second.</para>
755 /// <para></para>
756 /// </param>
757 /// <returns>
758 /// <para>The bool</para>
759 /// <para></para>
760 /// </returns>
761 [MethodImpl(MethodImplOptions.AggressiveInlining)]
762 protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
    ↪ second) < 0;

```

```

763     /// <summary>
764     /// <para>
765     /// Determines whether this instance less or equal than.
766     /// </para>
767     /// <para></para>
768     /// </summary>
769     /// <param name="first">
770     /// <para>The first.</para>
771     /// <para></para>
772     /// </param>
773     /// <param name="second">
774     /// <para>The second.</para>
775     /// <para></para>
776     /// </param>
777     /// <returns>
778     /// <para>The bool</para>
779     /// <para></para>
780     /// </returns>
781     [MethodImpl(MethodImplOptions.AggressiveInlining)]
782     protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
783         ↪ _comparer.Compare(first, second) <= 0;
784
785     /// <summary>
786     /// <para>
787     /// Determines whether this instance greater than.
788     /// </para>
789     /// <para></para>
790     /// </summary>
791     /// <param name="first">
792     /// <para>The first.</para>
793     /// <para></para>
794     /// </param>
795     /// <param name="second">
796     /// <para>The second.</para>
797     /// <para></para>
798     /// </param>
799     /// <returns>
800     /// <para>The bool</para>
801     /// <para></para>
802     /// </returns>
803     [MethodImpl(MethodImplOptions.AggressiveInlining)]
804     protected virtual bool GreaterThan(TLink first, TLink second) =>
805         ↪ _comparer.Compare(first, second) > 0;
806
807     /// <summary>
808     /// <para>
809     /// Determines whether this instance greater or equal than.
810     /// </para>
811     /// <para></para>
812     /// </summary>
813     /// <param name="first">
814     /// <para>The first.</para>
815     /// <para></para>
816     /// </param>
817     /// <param name="second">
818     /// <para>The second.</para>
819     /// <para></para>
820     /// </param>
821     /// <returns>
822     /// <para>The bool</para>
823     /// <para></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     /// <para></para>
834     /// </summary>
835     /// <param name="value">
836     /// <para>The value.</para>
837     /// <para></para>
838     /// </param>
839     /// <returns>
840     /// <para>The long</para>

```

```

838     /// <para></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     /// <para></para>
849     /// </summary>
850     /// <param name="value">
851     /// <para>The value.</para>
852     /// </param>
853     /// <returns>
854     /// <para>The link</para>
855     /// <para></para>
856     /// </returns>
857     [MethodImpl(MethodImplOptions.AggressiveInlining)]
858     protected virtual TLink ConvertToAddress(long value) =>
859         ↪ _int64ToAddressConverter.Convert(value);
860
861     /// <summary>
862     /// <para>
863     /// Adds the first.
864     /// </para>
865     /// <para></para>
866     /// </summary>
867     /// <param name="first">
868     /// <para>The first.</para>
869     /// </param>
870     /// <param name="second">
871     /// <para>The second.</para>
872     /// </param>
873     /// <returns>
874     /// <para>The link</para>
875     /// <para></para>
876     /// </returns>
877     [MethodImpl(MethodImplOptions.AggressiveInlining)]
878     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
879         ↪ second);
880
881     /// <summary>
882     /// <para>
883     /// Subtracts the first.
884     /// </para>
885     /// <para></para>
886     /// </summary>
887     /// <param name="first">
888     /// <para>The first.</para>
889     /// </param>
890     /// <param name="second">
891     /// <para>The second.</para>
892     /// </param>
893     /// <returns>
894     /// <para>The link</para>
895     /// <para></para>
896     /// </returns>
897     [MethodImpl(MethodImplOptions.AggressiveInlining)]
898     protected virtual TLink Subtract(TLink first, TLink second) =>
899         ↪ Arithmetic<TLink>.Subtract(first, second);
900
901     /// <summary>
902     /// <para>
903     /// Increments the link.
904     /// </para>
905     /// <para></para>
906     /// </summary>
907     /// <param name="link">
908     /// <para>The link.</para>
909     /// </param>
910     /// </summary>
911     /// </param>

```

```

912     /// <returns>
913     /// <para>The link</para>
914     /// <para></para>
915     /// </returns>
916     [MethodImpl(MethodImplOptions.AggressiveInlining)]
917     protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
918
919     /// <summary>
920     /// <para>
921     /// Decrements the link.
922     /// </para>
923     /// <para></para>
924     /// </summary>
925     /// <param name="link">
926     /// <para>The link.</para>
927     /// <para></para>
928     /// </param>
929     /// <returns>
930     /// <para>The link</para>
931     /// <para></para>
932     /// </returns>
933     [MethodImpl(MethodImplOptions.AggressiveInlining)]
934     protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
935
936     #region Disposable
937
938     /// <summary>
939     /// <para>
940     /// Gets the allow multiple dispose calls value.
941     /// </para>
942     /// <para></para>
943     /// </summary>
944     protected override bool AllowMultipleDisposeCalls
945     {
946         [MethodImpl(MethodImplOptions.AggressiveInlining)]
947         get => true;
948     }
949
950     /// <summary>
951     /// <para>
952     /// Disposes the manual.
953     /// </para>
954     /// <para></para>
955     /// </summary>
956     /// <param name="manual">
957     /// <para>The manual.</para>
958     /// <para></para>
959     /// </param>
960     /// <param name="wasDisposed">
961     /// <para>The was disposed.</para>
962     /// <para></para>
963     /// </param>
964     [MethodImpl(MethodImplOptions.AggressiveInlining)]
965     protected override void Dispose(bool manual, bool wasDisposed)
966     {
967         if (!wasDisposed)
968         {
969             ResetPointers();
970             _memory.DisposeIfPossible();
971         }
972     }
973
974     #endregion
975 }
976 }

```

1.88 ./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> :
    ↳ AbsoluteCircularDoublyLinkedListMethods<TLink>, ILinksListMethods<TLink>
19  {
20      private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
    ↳ UncheckedConverter<TLink, long>.Default;
21
22      private readonly byte* _links;
23      private readonly byte* _header;
24
25      /// <summary>
26      /// <para>
27      /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
28      /// </para>
29      /// <para></para>
30      /// </summary>
31      /// <param name="links">
32      /// <para>A links.</para>
33      /// <para></para>
34      /// </param>
35      /// <param name="header">
36      /// <para>A header.</para>
37      /// <para></para>
38      /// </param>
39      [MethodImpl(MethodImplOptions.AggressiveInlining)]
40      public UnusedLinksListMethods(byte* links, byte* header)
41      {
42          _links = links;
43          _header = header;
44      }
45
46      /// <summary>
47      /// <para>
48      /// Gets the header reference.
49      /// </para>
50      /// <para></para>
51      /// </summary>
52      /// <returns>
53      /// <para>A ref links header of t link</para>
54      /// <para></para>
55      /// </returns>
56      [MethodImpl(MethodImplOptions.AggressiveInlining)]
57      protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↳ 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
    ↳ AsRef<RawLink<TLink>>(_links + (RawLink<TLink>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));
75
76      /// <summary>
77      /// <para>
78      /// Gets the first.
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 GetFirst() => GetHeaderReference().FirstFreeLink;
88
89 /// <summary>
90 /// <para>
91 /// Gets the last.
92 /// </para>
93 /// <para></para>
94 /// </summary>
95 /// <returns>
96 /// <para>The link</para>
97 /// <para></para>
98 /// </returns>
99 [MethodImpl(MethodImplOptions.AggressiveInlining)]
100 protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
101
102 /// <summary>
103 /// <para>
104 /// Gets the previous using the specified element.
105 /// </para>
106 /// <para></para>
107 /// </summary>
108 /// <param name="element">
109 /// <para>The element.</para>
110 /// <para></para>
111 /// </param>
112 /// <returns>
113 /// <para>The link</para>
114 /// <para></para>
115 /// </returns>
116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 protected override TLink GetPrevious(TLink element) => GetLinkReference(element).Source;
118
119 /// <summary>
120 /// <para>
121 /// Gets the next using the specified element.
122 /// </para>
123 /// <para></para>
124 /// </summary>
125 /// <param name="element">
126 /// <para>The element.</para>
127 /// <para></para>
128 /// </param>
129 /// <returns>
130 /// <para>The link</para>
131 /// <para></para>
132 /// </returns>
133 [MethodImpl(MethodImplOptions.AggressiveInlining)]
134 protected override TLink GetNext(TLink element) => GetLinkReference(element).Target;
135
136 /// <summary>
137 /// <para>
138 /// Gets the size.
139 /// </para>
140 /// <para></para>
141 /// </summary>
142 /// <returns>
143 /// <para>The link</para>
144 /// <para></para>
145 /// </returns>
146 [MethodImpl(MethodImplOptions.AggressiveInlining)]
147 protected override TLink GetSize() => GetHeaderReference().FreeLinks;
148
149 /// <summary>
150 /// <para>
151 /// Sets the first using the specified element.
152 /// </para>
153 /// <para></para>
154 /// </summary>
155 /// <param name="element">
156 /// <para>The element.</para>
157 /// <para></para>
158 /// </param>
159 [MethodImpl(MethodImplOptions.AggressiveInlining)]
160 protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
    ↪ element;
161
162 /// <summary>

```

```

163     /// <para>
164     /// Sets the last using the specified element.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="element">
169     /// <para>The element.</para>
170     /// <para></para>
171     /// </param>
172     [MethodImpl(MethodImplOptions.AggressiveInlining)]
173     protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
        ↪ element;

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) =>
190     ↪ GetLinkReference(element).Source = previous;

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     /// <summary>
209     /// <para>
210     /// Sets the size using the specified size.
211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <param name="size">
215     /// <para>The size.</para>
216     /// <para></para>
217     /// </param>
218     [MethodImpl(MethodImplOptions.AggressiveInlining)]
219     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
220 }
221 }
222 }

```

1.89 ./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      /// </summary>
26      public static readonly long SizeInBytes = Structure<RawLink<TLink>>.Size;
27
28      /// <summary>
29      /// <para>
30      /// The source.
31      /// </para>
32      /// <para></para>
33      /// </summary>
34      public TLink Source;
35      /// <summary>
36      /// <para>
37      /// The target.
38      /// </para>
39      /// <para></para>
40      /// </summary>
41      public TLink Target;
42      /// <summary>
43      /// <para>
44      /// The left as source.
45      /// </para>
46      /// <para></para>
47      /// </summary>
48      public TLink LeftAsSource;
49      /// <summary>
50      /// <para>
51      /// The right as source.
52      /// </para>
53      /// <para></para>
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 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 links recursionless size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{uint}"/>
15     public unsafe abstract class UInt32LinksRecursionlessSizeBalancedTreeMethodsBase :
        => LinksRecursionlessSizeBalancedTreeMethodsBase<uint>
16     {

```

```

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

```

```

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;

```

```

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 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.United.Generic;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.United.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 32 links size balanced tree methods base.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="LinksSizeBalancedTreeMethodsBase{uint}"/>
15    public unsafe abstract class UInt32LinksSizeBalancedTreeMethodsBase :
16    ↪ LinksSizeBalancedTreeMethodsBase<uint>
17    {
18        /// <summary>
19        /// <para>
20        /// The links.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        protected new readonly RawLink<uint>* Links;
25
26        /// <summary>
27        /// <para>
28        /// The header.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        protected new readonly LinksHeader<uint>* Header;
33
34        /// <summary>
35        /// <para>
36        /// Initializes a new <see cref="UInt32LinksSizeBalancedTreeMethodsBase"/> instance.
37        /// </para>
38        /// <para></para>
39        /// </summary>
40        /// <param name="constants">
41        /// <para>A constants.</para>
42        /// <para></para>
43        /// </param>
44        /// <param name="links">
45        /// <para>A links.</para>
46        /// <para></para>
47        /// </param>
48        /// <param name="header">
49        /// <para>A header.</para>
50        /// <para></para>
51        /// </param>
52        protected UInt32LinksSizeBalancedTreeMethodsBase(LinksConstants<uint> constants,
53    ↪ RawLink<uint>* links, LinksHeader<uint>* header)
54        : base(constants, (byte*)links, (byte*)header)
55        {
56            Links = links;
57            Header = header;
58        }
59    }
60 }

```

```

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

```

```

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

```

```

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

```

```

288     /// <para>The uint</para>
289     /// <para></para>
290     /// </returns>
291     [MethodImpl(MethodImplOptions.AggressiveInlining)]
292     protected override uint Decrement(uint value) => --value;
293
294     /// <summary>
295     /// <para>
296     /// Adds the first.
297     /// </para>
298     /// <para></para>
299     /// </summary>
300     /// <param name="first">
301     /// <para>The first.</para>
302     /// <para></para>
303     /// </param>
304     /// <param name="second">
305     /// <para>The second.</para>
306     /// <para></para>
307     /// </param>
308     /// <returns>
309     /// <para>The uint</para>
310     /// <para></para>
311     /// </returns>
312     [MethodImpl(MethodImplOptions.AggressiveInlining)]
313     protected override uint Add(uint first, uint second) => first + second;
314
315     /// <summary>
316     /// <para>
317     /// Subtracts the first.
318     /// </para>
319     /// <para></para>
320     /// </summary>
321     /// <param name="first">
322     /// <para>The first.</para>
323     /// <para></para>
324     /// </param>
325     /// <param name="second">
326     /// <para>The second.</para>
327     /// <para></para>
328     /// </param>
329     /// <returns>
330     /// <para>The uint</para>
331     /// <para></para>
332     /// </returns>
333     [MethodImpl(MethodImplOptions.AggressiveInlining)]
334     protected override uint Subtract(uint first, uint second) => first - second;
335
336     /// <summary>
337     /// <para>
338     /// Determines whether this instance first is to the left of second.
339     /// </para>
340     /// <para></para>
341     /// </summary>
342     /// <param name="first">
343     /// <para>The first.</para>
344     /// <para></para>
345     /// </param>
346     /// <param name="second">
347     /// <para>The second.</para>
348     /// <para></para>
349     /// </param>
350     /// <returns>
351     /// <para>The bool</para>
352     /// <para></para>
353     /// </returns>
354     [MethodImpl(MethodImplOptions.AggressiveInlining)]
355     protected override bool FirstIsToLeftOfSecond(uint first, uint second)
356     {
357         ref var firstLink = ref Links[first];
358         ref var secondLink = ref Links[second];
359         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
360             ↪ secondLink.Source, secondLink.Target);
361     }
362
363     /// <summary>
364     /// <para>
365     /// Determines whether this instance first is to the right of second.

```

```

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 }

```

1.92 ./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>

```

```

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 UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
        ↳ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
        ↳ header) { }
35
36    /// <summary>
37    /// <para>
38    /// Gets the left reference using the specified node.
39    /// </para>
40    /// <para></para>
41    /// </summary>
42    /// <param name="node">
43    /// <para>The node.</para>
44    /// <para></para>
45    /// </param>
46    /// <returns>
47    /// <para>The ref uint</para>
48    /// <para></para>
49    /// </returns>
50    [MethodImpl(MethodImplOptions.AggressiveInlining)]
51    protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
52
53    /// <summary>
54    /// <para>
55    /// Gets the right reference using the specified node.
56    /// </para>
57    /// <para></para>
58    /// </summary>
59    /// <param name="node">
60    /// <para>The node.</para>
61    /// <para></para>
62    /// </param>
63    /// <returns>
64    /// <para>The ref uint</para>
65    /// <para></para>
66    /// </returns>
67    [MethodImpl(MethodImplOptions.AggressiveInlining)]
68    protected override ref uint GetRightReference(uint node) => ref
        ↳ Links[node].RightAsSource;
69
70    /// <summary>
71    /// <para>
72    /// Gets the left using the specified node.
73    /// </para>
74    /// <para></para>
75    /// </summary>
76    /// <param name="node">
77    /// <para>The node.</para>
78    /// <para></para>
79    /// </param>
80    /// <returns>
81    /// <para>The uint</para>
82    /// <para></para>
83    /// </returns>
84    [MethodImpl(MethodImplOptions.AggressiveInlining)]
85    protected override uint GetLeft(uint node) => Links[node].LeftAsSource;
86
87    /// <summary>
88    /// <para>
89    /// Gets the right using the specified node.
90    /// </para>
91    /// <para></para>
92    /// </summary>
93    /// <param name="node">
94    /// <para>The node.</para>

```



```

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

```

```

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>

```

```

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

```

1.93 ./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 :
        ↪ UInt32LinksSizeBalancedTreeMethodsBase
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="UInt32LinksSourcesSizeBalancedTreeMethods"/> 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 UInt32LinksSourcesSizeBalancedTreeMethods(LinksConstants<uint> constants,
        ↪ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
35
36        /// <summary>
37        /// <para>
38        /// Gets the left reference using the specified node.

```

```

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

```

```

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

```

```

193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>The uint</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override uint GetBasePartValue(uint link) => Links[link].Source;
201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance first is to the left of second.
205     /// </para>
206     /// <para></para>
207     /// </summary>
208     /// <param name="firstSource">
209     /// <para>The first source.</para>
210     /// <para></para>
211     /// </param>
212     /// <param name="firstTarget">
213     /// <para>The first target.</para>
214     /// <para></para>
215     /// </param>
216     /// <param name="secondSource">
217     /// <para>The second source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool 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>

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links targets recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods :
15         ↳ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↳ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// <para></para>
35         /// </param>
36         public UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
37             ↳ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
38             ↳ header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="node">
47         /// <para>The node.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The ref uint</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsTarget;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>
67         /// <returns>
68         /// <para>The ref uint</para>
69         /// <para></para>
70         /// </returns>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         protected override ref uint GetRightReference(uint node) => ref Links[node].RightAsTarget;
73     }
74 }

```

```

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

```



```

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

```

```

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

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

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(uint node)
273     {
274         ref var link = ref Links[node];
275         link.LeftAsTarget = 0U;
276         link.RightAsTarget = 0U;
277         link.SizeAsTarget = 0U;
278     }
279 }
280 }
281

```

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

```
1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links targets size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksTargetsSizeBalancedTreeMethods :
15         ↳ UInt32LinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt32LinksTargetsSizeBalancedTreeMethods(LinksConstants<uint> constants,
36             ↳ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
37
38         /// <summary>
39         /// <para>
40         /// Gets the left reference using the specified node.
41         /// </para>
42         /// <para></para>
43         /// </summary>
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].LeftAsTarget;
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].RightAsTarget;
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].LeftAsTarget;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The uint</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override uint GetRight(uint node) => Links[node].RightAsTarget;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// <para></para>
117    /// </param>
118    [MethodImpl(MethodImplOptions.AggressiveInlining)]
119    protected override void SetLeft(uint node, uint left) => Links[node].LeftAsTarget = left;
120
121    /// <summary>
122    /// <para>
123    /// Sets the right using the specified node.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="node">
128    /// <para>The node.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="right">
132    /// <para>The right.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]
136    protected override void SetRight(uint node, uint right) => Links[node].RightAsTarget =
        right;
137
138    /// <summary>
139    /// <para>
140    /// Gets the size using the specified node.
141    /// </para>
142    /// <para></para>
143    /// </summary>
144    /// <param name="node">
145    /// <para>The node.</para>
146    /// <para></para>
147    /// </param>
148    /// <returns>
149    /// <para>The uint</para>
150    /// <para></para>
151    /// </returns>

```

```

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

```

```

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

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Memory;
4  using Platform.Singletons;
5  using Platform.Data.Doublets.Memory.United.Generic;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     ↪ organizing the storage of links with addresses represented as <see cref="uint" />.</para>
14     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
15     ↪ размером, для организации хранения связей с адресами представленными в виде <see
16     ↪ cref="uint" />.</para>
17     /// </summary>
18     public unsafe class UInt32UnitedMemoryLinks : UnitedMemoryLinksBase<uint>
19     {

```

```

17 private readonly Func<ILinksTreeMethods<uint>> _createSourceTreeMethods;
18 private readonly Func<ILinksTreeMethods<uint>> _createTargetTreeMethods;
19 private LinksHeader<uint>* _header;
20 private RawLink<uint>* _links;
21
22 /// <summary>
23 /// <para>
24 /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
25 /// </para>
26 /// <para></para>
27 /// </summary>
28 /// <param name="address">
29 /// <para>A address.</para>
30 /// <para></para>
31 /// </param>
32 [MethodImpl(MethodImplOptions.AggressiveInlining)]
33 public UInt32UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
34
35 /// <summary>
36 /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
37   ↳ минимальным шагом расширения базы данных.
38 /// </summary>
39 /// <param name="address">Полный путь к файлу базы данных.</param>
40 /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
41   ↳ байтах.</param>
42 [MethodImpl(MethodImplOptions.AggressiveInlining)]
43 public UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
44   ↳ FileMappedResizableDirectMemory(address, memoryReservationStep),
45   ↳ memoryReservationStep) { }
46
47 /// <summary>
48 /// <para>
49 /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
50 /// </para>
51 /// <para></para>
52 /// </summary>
53 /// <param name="memory">
54 /// <para>A memory.</para>
55 /// <para></para>
56 /// </param>
57 [MethodImpl(MethodImplOptions.AggressiveInlining)]
58 public UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
59   ↳ DefaultLinksSizeStep) { }
60
61 /// <summary>
62 /// <para>
63 /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
64 /// </para>
65 /// <para></para>
66 /// </summary>
67 /// <param name="memory">
68 /// <para>A memory.</para>
69 /// <para></para>
70 /// </param>
71 /// <param name="memoryReservationStep">
72 /// <para>A memory reservation step.</para>
73 /// <para></para>
74 /// </param>
75 [MethodImpl(MethodImplOptions.AggressiveInlining)]
76 public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
77   ↳ memoryReservationStep) : this(memory, memoryReservationStep,
78   ↳ Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }
79
80 /// <summary>
81 /// <para>
82 /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
83 /// </para>
84 /// <para></para>
85 /// </summary>
86 /// <param name="memory">
87 /// <para>A memory.</para>
88 /// <para></para>
89 /// </param>
90 /// <param name="memoryReservationStep">
91 /// <para>A memory reservation step.</para>
92 /// <para></para>
93 /// </param>
94 /// <param name="constants">

```

```

88  /// <para>A constants.</para>
89  /// <para></para>
90  /// </param>
91  /// <param name="indexTreeType">
92  /// <para>A index tree type.</para>
93  /// <para></para>
94  /// </param>
95  [MethodImpl(MethodImplOptions.AggressiveInlining)]
96  public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
    ↳ memoryReservationStep, LinksConstants<uint> constants, IndexTreeType indexTreeType)
    ↳ : base(memory, memoryReservationStep, constants)
97  {
98      if (indexTreeType == IndexTreeType.SizeBalancedTree)
99      {
100          _createSourceTreeMethods = () => new
            ↳ UInt32LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
101          _createTargetTreeMethods = () => new
            ↳ UInt32LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
102      }
103      else
104      {
105          _createSourceTreeMethods = () => new
            ↳ UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
            ↳ _header);
106          _createTargetTreeMethods = () => new
            ↳ UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
            ↳ _header);
107      }
108      Init(memory, memoryReservationStep);
109  }
110
111  /// <summary>
112  /// <para>
113  /// Sets the pointers using the specified memory.
114  /// </para>
115  /// <para></para>
116  /// </summary>
117  /// <param name="memory">
118  /// <para>The memory.</para>
119  /// <para></para>
120  /// </param>
121  [MethodImpl(MethodImplOptions.AggressiveInlining)]
122  protected override void SetPointers(IResizableDirectMemory memory)
123  {
124      _header = (LinksHeader<uint>*)memory.Pointer;
125      _links = (RawLink<uint>*)memory.Pointer;
126      SourcesTreeMethods = _createSourceTreeMethods();
127      TargetsTreeMethods = _createTargetTreeMethods();
128      UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_links, _header);
129  }
130
131  /// <summary>
132  /// <para>
133  /// Resets the pointers.
134  /// </para>
135  /// <para></para>
136  /// </summary>
137  [MethodImpl(MethodImplOptions.AggressiveInlining)]
138  protected override void ResetPointers()
139  {
140      base.ResetPointers();
141      _links = null;
142      _header = null;
143  }
144
145  /// <summary>
146  /// <para>
147  /// Gets the header reference.
148  /// </para>
149  /// <para></para>
150  /// </summary>
151  /// <returns>
152  /// <para>A ref links header of uint</para>
153  /// <para></para>
154  /// </returns>
155  [MethodImpl(MethodImplOptions.AggressiveInlining)]
156  protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
157

```



```

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

```

```

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

```

```

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

```

```

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

```

1.97 ./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 }

```

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

```

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

```

```

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

```

```

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

```

```

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

```



```

286    /// <para></para>
287    /// </param>
288    /// <returns>
289    /// <para>The ulong</para>
290    /// <para></para>
291    /// </returns>
292    [MethodImpl(MethodImplOptions.AggressiveInlining)]
293    protected override ulong Decrement(ulong value) => --value;
294
295    /// <summary>
296    /// <para>
297    /// Adds the first.
298    /// </para>
299    /// <para></para>
300    /// </summary>
301    /// <param name="first">
302    /// <para>The first.</para>
303    /// <para></para>
304    /// </param>
305    /// <param name="second">
306    /// <para>The second.</para>
307    /// <para></para>
308    /// </param>
309    /// <returns>
310    /// <para>The ulong</para>
311    /// <para></para>
312    /// </returns>
313    [MethodImpl(MethodImplOptions.AggressiveInlining)]
314    protected override ulong Add(ulong first, ulong second) => first + second;
315
316    /// <summary>
317    /// <para>
318    /// Subtracts the first.
319    /// </para>
320    /// <para></para>
321    /// </summary>
322    /// <param name="first">
323    /// <para>The first.</para>
324    /// <para></para>
325    /// </param>
326    /// <param name="second">
327    /// <para>The second.</para>
328    /// <para></para>
329    /// </param>
330    /// <returns>
331    /// <para>The ulong</para>
332    /// <para></para>
333    /// </returns>
334    [MethodImpl(MethodImplOptions.AggressiveInlining)]
335    protected override ulong Subtract(ulong first, ulong second) => first - second;
336
337    /// <summary>
338    /// <para>
339    /// Determines whether this instance first is to the left of second.
340    /// </para>
341    /// <para></para>
342    /// </summary>
343    /// <param name="first">
344    /// <para>The first.</para>
345    /// <para></para>
346    /// </param>
347    /// <param name="second">
348    /// <para>The second.</para>
349    /// <para></para>
350    /// </param>
351    /// <returns>
352    /// <para>The bool</para>
353    /// <para></para>
354    /// </returns>
355    [MethodImpl(MethodImplOptions.AggressiveInlining)]
356    protected override bool FirstIsToTheLeftOfSecond(ulong first, ulong second)
357    {
358        ref var firstLink = ref Links[first];
359        ref var secondLink = ref Links[second];
360        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
361            ↪ secondLink.Source, secondLink.Target);
362    }

```

```

363     /// <summary>
364     /// <para>
365     /// Determines whether this instance first is to the right of second.
366     /// </para>
367     /// <para></para>
368     /// </summary>
369     /// <param name="first">
370     /// <para>The first.</para>
371     /// <para></para>
372     /// </param>
373     /// <param name="second">
374     /// <para>The second.</para>
375     /// <para></para>
376     /// </param>
377     /// <returns>
378     /// <para>The bool</para>
379     /// <para></para>
380     /// </returns>
381     [MethodImpl(MethodImplOptions.AggressiveInlining)]
382     protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
383     {
384         ref var firstLink = ref Links[first];
385         ref var secondLink = ref Links[second];
386         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
387             ↪ secondLink.Source, secondLink.Target);
388     }
389     /// <summary>
390     /// <para>
391     /// Gets the size value using the specified value.
392     /// </para>
393     /// <para></para>
394     /// </summary>
395     /// <param name="value">
396     /// <para>The value.</para>
397     /// <para></para>
398     /// </param>
399     /// <returns>
400     /// <para>The ulong</para>
401     /// <para></para>
402     /// </returns>
403     [MethodImpl(MethodImplOptions.AggressiveInlining)]
404     protected override ulong GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;
405
406     /// <summary>
407     /// <para>
408     /// Sets the size value using the specified stored value.
409     /// </para>
410     /// <para></para>
411     /// </summary>
412     /// <param name="storedValue">
413     /// <para>The stored value.</para>
414     /// <para></para>
415     /// </param>
416     /// <param name="size">
417     /// <para>The size.</para>
418     /// <para></para>
419     /// </param>
420     [MethodImpl(MethodImplOptions.AggressiveInlining)]
421     protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
422         ↪ storedValue & 31UL | (size & 134217727UL) << 5;
423
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance get left is child value.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="value">
431     /// <para>The value.</para>
432     /// <para></para>
433     /// </param>
434     /// <returns>
435     /// <para>The bool</para>
436     /// <para></para>
437     /// </returns>
438     [MethodImpl(MethodImplOptions.AggressiveInlining)]
439     protected override bool GetLeftIsChildValue(ulong value) => (value & 16UL) >> 4 == 1UL;

```

```

439
440 /// <summary>
441 /// <para>
442 /// Sets the left is child value using the specified stored value.
443 /// </para>
444 /// <para></para>
445 /// </summary>
446 /// <param name="storedValue">
447 /// <para>The stored value.</para>
448 /// <para></para>
449 /// </param>
450 /// <param name="value">
451 /// <para>The value.</para>
452 /// <para></para>
453 /// </param>
454 [MethodImpl(MethodImplOptions.AggressiveInlining)]
455 protected override void SetLeftIsChildValue(ref ulong storedValue, bool value) =>
456     ↪ storedValue = storedValue & 4294967279UL | (As<bool, byte>(ref value) & 1UL) << 4;
457
458 /// <summary>
459 /// <para>
460 /// Determines whether this instance get right is child value.
461 /// </para>
462 /// <para></para>
463 /// </summary>
464 /// <param name="value">
465 /// <para>The value.</para>
466 /// <para></para>
467 /// </param>
468 /// <returns>
469 /// <para>The bool</para>
470 /// <para></para>
471 /// </returns>
472 [MethodImpl(MethodImplOptions.AggressiveInlining)]
473 protected override bool GetRightIsChildValue(ulong value) => (value & 8UL) >> 3 == 1UL;
474
475 /// <summary>
476 /// <para>
477 /// Sets the right is child value using the specified stored value.
478 /// </para>
479 /// <para></para>
480 /// </summary>
481 /// <param name="storedValue">
482 /// <para>The stored value.</para>
483 /// <para></para>
484 /// </param>
485 /// <param name="value">
486 /// <para>The value.</para>
487 /// <para></para>
488 /// </param>
489 [MethodImpl(MethodImplOptions.AggressiveInlining)]
490 protected override void SetRightIsChildValue(ref ulong storedValue, bool value) =>
491     ↪ storedValue = storedValue & 4294967287UL | (As<bool, byte>(ref value) & 1UL) << 3;
492
493 /// <summary>
494 /// <para>
495 /// Gets the balance value using the specified value.
496 /// </para>
497 /// <para></para>
498 /// </summary>
499 /// <param name="value">
500 /// <para>The value.</para>
501 /// <para></para>
502 /// </param>
503 /// <returns>
504 /// <para>The sbyte</para>
505 /// <para></para>
506 /// </returns>
507 [MethodImpl(MethodImplOptions.AggressiveInlining)]
508 protected override sbyte GetBalanceValue(ulong value) => unchecked((sbyte)(value & 7UL |
509     ↪ 0xF8UL * ((value & 4UL) >> 2))); // if negative, then continue ones to the end of
510     ↪ sbyte
511
512 /// <summary>
513 /// <para>
514 /// Sets the balance value using the specified stored value.
515 /// </para>
516 /// <para></para>

```

```

513     /// </summary>
514     /// <param name="storedValue">
515     /// <para>The stored value.</para>
516     /// <para></para>
517     /// </param>
518     /// <param name="value">
519     /// <para>The value.</para>
520     /// <para></para>
521     /// </param>
522     [MethodImpl(MethodImplOptions.AggressiveInlining)]
523     protected override void SetBalanceValue(ref ulong storedValue, sbyte value) =>
524     ↪ storedValue = unchecked(storedValue & 4294967288UL | (ulong)((byte)value >> 5 & 4 |
525     ↪ value & 3) & 7UL);
526
527     /// <summary>
528     /// <para>
529     /// Gets the header reference.
530     /// </para>
531     /// <para></para>
532     /// </summary>
533     /// <returns>
534     /// <para>A ref links header of ulong</para>
535     /// <para></para>
536     /// </returns>
537     [MethodImpl(MethodImplOptions.AggressiveInlining)]
538     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
539
540     /// <summary>
541     /// <para>
542     /// Gets the link reference using the specified link.
543     /// </para>
544     /// <para></para>
545     /// </summary>
546     /// <param name="link">
547     /// <para>The link.</para>
548     /// <para></para>
549     /// </param>
550     /// <returns>
551     /// <para>A ref raw link of ulong</para>
552     /// <para></para>
553     /// </returns>
554     [MethodImpl(MethodImplOptions.AggressiveInlining)]
555     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
556 }
557 }

```

1.99 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethods

```

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
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="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
35     /// instance.
36     /// </para>
37     /// </summary>
38     /// <param name="constants">
39     /// <para>A constants.</para>
40     /// </param>
41     /// <param name="links">
42     /// <para>A links.</para>
43     /// </param>
44     /// <param name="header">
45     /// <para>A header.</para>
46     /// </param>
47     protected UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
48     → constants, 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() => 0UL;
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 == 0UL;
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 }

```

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

```

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

```

```

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

```

```

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

```

```

225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <param name="value">
229     /// <para>The value.</para>
230     /// <para></para>
231     /// </param>
232     /// <returns>
233     /// <para>The bool</para>
234     /// <para></para>
235     /// </returns>
236     [MethodImpl(MethodImplOptions.AggressiveInlining)]
237     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↪    for ulong

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

```

```

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

```

```

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 }

```

1.101 ./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>

```

```

34 public UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
    ↳ { }
35
36 /// <summary>
37 /// <para>
38 /// Gets the left reference using the specified node.
39 /// </para>
40 /// <para></para>
41 /// </summary>
42 /// <param name="node">
43 /// <para>The node.</para>
44 /// <para></para>
45 /// </param>
46 /// <returns>
47 /// <para>The ref ulong</para>
48 /// <para></para>
49 /// </returns>
50 [MethodImpl(MethodImplOptions.AggressiveInlining)]
51 protected override ref ulong GetLeftReference(ulong node) => ref
    ↳ 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) => 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) =>
188         ↪ GetLeftIsChildValue(Links[node].SizeAsSource);
189
190     ///[MethodImpl(MethodImplOptions.AggressiveInlining)]
191     ///protected override bool GetLeftIsChild(ulong node) => IsChild(node, GetLeft(node));
192
193     /// <summary>
194     /// <para>
195     /// Sets the left is child using the specified node.
196     /// </para>
197     /// </summary>
198     /// <param name="node">
199     /// <para>The node.</para>
200     /// <para></para>
201     /// </param>
202     /// <param name="value">
203     /// <para>The value.</para>
204     /// <para></para>
205     /// </param>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override void SetLeftIsChild(ulong node, bool value) =>
208         ↪ SetLeftIsChildValue(ref Links[node].SizeAsSource, value);
209
210     /// <summary>
211     /// <para>
212     /// Determines whether this instance get right is child.
213     /// </para>
214     /// </summary>
215     /// <param name="node">
216     /// <para>The node.</para>
217     /// <para></para>
218     /// </param>
219     /// <returns>
220     /// <para>The bool</para>
221     /// <para></para>
222     /// </returns>
223     [MethodImpl(MethodImplOptions.AggressiveInlining)]
224     protected override bool GetRightIsChild(ulong node) =>
225         ↪ GetRightIsChildValue(Links[node].SizeAsSource);
226
227     ///[MethodImpl(MethodImplOptions.AggressiveInlining)]
228     ///protected override bool GetRightIsChild(ulong node) => IsChild(node, GetRight(node));
229
230     /// <summary>
231     /// <para>
232     /// Sets the right is child using the specified node.
233     /// </para>
234     /// </summary>
235     /// <param name="node">
236     /// <para>The node.</para>
237     /// <para></para>
238     /// </param>
239     /// <param name="value">
240     /// <para>The value.</para>
241     /// <para></para>
242     /// </param>
243     [MethodImpl(MethodImplOptions.AggressiveInlining)]
244     protected override void SetRightIsChild(ulong node, bool value) =>
245         ↪ SetRightIsChildValue(ref Links[node].SizeAsSource, value);
246
247     /// <summary>
248     /// <para>
249     /// Gets the balance using the specified node.
250     /// </para>
251     /// </summary>
252     /// <param name="node">
253     /// <para>The node.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>

```

```

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

```

```

333     /// <para>The bool</para>
334     /// <para></para>
335     /// </returns>
336     [MethodImpl(MethodImplOptions.AggressiveInlining)]
337     protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
338     ↪     ulong secondSource, ulong secondTarget)
339     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
340     ↪     secondTarget);
341
342     /// <summary>
343     /// <para>
344     /// Determines whether this instance first is to the right of second.
345     /// </para>
346     /// <para></para>
347     /// </summary>
348     /// <param name="firstSource">
349     /// <para>The first source.</para>
350     /// <para></para>
351     /// </param>
352     /// <param name="firstTarget">
353     /// <para>The first target.</para>
354     /// <para></para>
355     /// </param>
356     /// <param name="secondSource">
357     /// <para>The second source.</para>
358     /// <para></para>
359     /// </param>
360     /// <param name="secondTarget">
361     /// <para>The second target.</para>
362     /// <para></para>
363     /// </param>
364     /// <returns>
365     /// <para>The bool</para>
366     /// <para></para>
367     /// </returns>
368     [MethodImpl(MethodImplOptions.AggressiveInlining)]
369     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
370     ↪     ulong secondSource, ulong secondTarget)
371     => firstSource > secondSource || (firstSource == secondSource && firstTarget >
372     ↪     secondTarget);
373
374     /// <summary>
375     /// <para>
376     /// Clears the node using the specified node.
377     /// </para>
378     /// <para></para>
379     /// </summary>
380     /// <param name="node">
381     /// <para>The node.</para>
382     /// <para></para>
383     /// </param>
384     [MethodImpl(MethodImplOptions.AggressiveInlining)]
385     protected override void ClearNode(ulong node)
386     {
387         ref var link = ref Links[node];
388         link.LeftAsSource = OUL;
389         link.RightAsSource = OUL;
390         link.SizeAsSource = OUL;
391     }
392 }

```

1.102 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethodsBase

```

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

```

```

15 {
16     /// <summary>
17     /// <para>
18     /// Initializes a new <see
19     ↪ cref="UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
36     ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
37     ↪ 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 ulong</para>
51     /// <para></para>
52     /// </returns>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     protected override ref ulong GetLeftReference(ulong node) => ref
55     ↪ Links[node].LeftAsSource;
56
57     /// <summary>
58     /// <para>
59     /// Gets the right reference using the specified node.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <param name="node">
64     /// <para>The node.</para>
65     /// <para></para>
66     /// </param>
67     /// <returns>
68     /// <para>The ref ulong</para>
69     /// <para></para>
70     /// </returns>
71     [MethodImpl(MethodImplOptions.AggressiveInlining)]
72     protected override ref ulong GetRightReference(ulong node) => ref
73     ↪ Links[node].RightAsSource;
74
75     /// <summary>
76     /// <para>
77     /// Gets the left using the specified node.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <param name="node">
82     /// <para>The node.</para>
83     /// <para></para>
84     /// </param>
85     /// <returns>
86     /// <para>The ulong</para>
87     /// <para></para>
88     /// </returns>
89     [MethodImpl(MethodImplOptions.AggressiveInlining)]
90     protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
91

```

```

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

```

```

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

```

```

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

```

1.103 ./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 :
        ↪     UInt64LinksSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="UInt64LinksSourcesSizeBalancedTreeMethods"/> 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 UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
    ↳ { }

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

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

69
70  /// <summary>
71  /// <para>
72  /// Gets the left using the specified node.
73  /// </para>
74  /// <para></para>
75  /// </summary>
76  /// <param name="node">
77  /// <para>The node.</para>
78  /// <para></para>
79  /// </param>
80  /// <returns>
81  /// <para>The ulong</para>
82  /// <para></para>
83  /// </returns>
84  [MethodImpl(MethodImplOptions.AggressiveInlining)]
85  protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
86
87  /// <summary>
88  /// <para>
89  /// Gets the right using the specified node.
90  /// </para>
91  /// <para></para>
92  /// </summary>
93  /// <param name="node">
94  /// <para>The node.</para>
95  /// <para></para>
96  /// </param>
97  /// <returns>
98  /// <para>The ulong</para>
99  /// <para></para>
100  /// </returns>
101  [MethodImpl(MethodImplOptions.AggressiveInlining)]
102  protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
103

```

```

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

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

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

171
172    /// <summary>
173    /// <para>
174    /// Gets the tree root.
175    /// </para>
176    /// <para></para>
177    /// </summary>
178    /// <returns>

```

```

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

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

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

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

```

```

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

```

1.104 ./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>

```

```

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

```

```

120
121     /// <summary>
122     /// <para>
123     /// Sets the right using the specified node.
124     /// </para>
125     /// <para></para>
126     /// </summary>
127     /// <param name="node">
128     /// <para>The node.</para>
129     /// <para></para>
130     /// </param>
131     /// <param name="right">
132     /// <para>The right.</para>
133     /// <para></para>
134     /// </param>
135     [MethodImpl(MethodImplOptions.AggressiveInlining)]
136     protected override void SetRight(ulong node, ulong right) => Links[node].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) =>
205         ↪ SetLeftIsChildValue(ref Links[node].SizeAsTarget, value);
206
207     /// <summary>
208     /// <para>
209     /// Determines whether this instance get right is child.
210     /// </para>
211     /// <para></para>
212     /// </summary>
213     /// <param name="node">
214     /// <para>The node.</para>
215     /// <para></para>
216     /// </param>
217     /// <returns>
218     /// <para>The bool</para>
219     /// <para></para>
220     /// </returns>
221     [MethodImpl(MethodImplOptions.AggressiveInlining)]
222     protected override bool GetRightIsChild(ulong node) =>
223         ↪ GetRightIsChildValue(Links[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(ulong node, bool value) =>
241         ↪ SetRightIsChildValue(ref Links[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(ulong node) =>
259         ↪ GetBalanceValue(Links[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>

```

```

269     /// <para></para>
270     /// </param>
271     [MethodImpl(MethodImplOptions.AggressiveInlining)]
272     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
    ↪ Links[node].SizeAsTarget, value);

273
274     /// <summary>
275     /// <para>
276     /// Gets the tree root.
277     /// </para>
278     /// <para></para>
279     /// </summary>
280     /// <returns>
281     /// <para>The ulong</para>
282     /// <para></para>
283     /// </returns>
284     [MethodImpl(MethodImplOptions.AggressiveInlining)]
285     protected override ulong GetTreeRoot() => Header->RootAsTarget;
286
287     /// <summary>
288     /// <para>
289     /// Gets the base part value using the specified link.
290     /// </para>
291     /// <para></para>
292     /// </summary>
293     /// <param name="link">
294     /// <para>The link.</para>
295     /// <para></para>
296     /// </param>
297     /// <returns>
298     /// <para>The ulong</para>
299     /// <para></para>
300     /// </returns>
301     [MethodImpl(MethodImplOptions.AggressiveInlining)]
302     protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
303
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,
    ↪ ulong secondSource, ulong secondTarget)
332     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
    ↪ secondSource);
333
334     /// <summary>
335     /// <para>
336     /// Determines whether this instance first is to the right of second.
337     /// </para>
338     /// <para></para>
339     /// </summary>
340     /// <param name="firstSource">
341     /// <para>The first source.</para>
342     /// <para></para>
343     /// </param>

```



```

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

```

1.105 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods

```

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>

```

```

33     /// </param>
34     public UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
    ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
    ↪ links, header) { }
35
36     /// <summary>
37     /// <para>
38     /// Gets the left reference using the specified node.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref ulong</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
    ↪ Links[node].LeftAsTarget;
52
53     /// <summary>
54     /// <para>
55     /// Gets the right reference using the specified node.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
    ↪ Links[node].RightAsTarget;
69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetLeft(ulong node) => Links[node].LeftAsTarget;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsTarget;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.

```

```

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

```

```

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

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 64 links targets size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt64LinksTargetsSizeBalancedTreeMethods :
15    ↳ UInt64LinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt64LinksTargetsSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// </para></para>
26        /// <param name="links">
27        /// <para>A links.</para>
28        /// </para></para>
29        /// <param name="header">
30        /// <para>A header.</para>
31        /// </para></para>
32        /// </param>
33        public UInt64LinksTargetsSizeBalancedTreeMethods(LinksConstants<ulong> constants,
34    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
35    ↳ { }
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 ulong</para>

```

```

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

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

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

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

```

```

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

```

```

199 [MethodImpl(MethodImplOptions.AggressiveInlining)]
200 protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
201
202 /// <summary>
203 /// <para>
204 /// Determines whether this instance first is to the left of second.
205 /// </para>
206 /// <para></para>
207 /// </summary>
208 /// <param name="firstSource">
209 /// <para>The first source.</para>
210 /// <para></para>
211 /// </param>
212 /// <param name="firstTarget">
213 /// <para>The first target.</para>
214 /// <para></para>
215 /// </param>
216 /// <param name="secondSource">
217 /// <para>The second source.</para>
218 /// <para></para>
219 /// </param>
220 /// <param name="secondTarget">
221 /// <para>The second target.</para>
222 /// <para></para>
223 /// </param>
224 /// <returns>
225 /// <para>The bool</para>
226 /// <para></para>
227 /// </returns>
228 [MethodImpl(MethodImplOptions.AggressiveInlining)]
229 protected override bool FirstIsToLeftOfSecond(ulong firstSource, ulong firstTarget,
230     ↪ ulong secondSource, ulong secondTarget)
231     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
232     ↪ secondSource);
233
234 /// <summary>
235 /// <para>
236 /// Determines whether this instance first is to the right of second.
237 /// </para>
238 /// <para></para>
239 /// </summary>
240 /// <param name="firstSource">
241 /// <para>The first source.</para>
242 /// <para></para>
243 /// </param>
244 /// <param name="firstTarget">
245 /// <para>The first target.</para>
246 /// <para></para>
247 /// </param>
248 /// <param name="secondSource">
249 /// <para>The second source.</para>
250 /// <para></para>
251 /// </param>
252 /// <param name="secondTarget">
253 /// <para>The second target.</para>
254 /// <para></para>
255 /// </param>
256 /// <returns>
257 /// <para>The bool</para>
258 /// <para></para>
259 /// </returns>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
262     ↪ ulong secondSource, ulong secondTarget)
263     => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
264     ↪ secondSource);
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.LeftAsTarget = OUL;
277         link.RightAsTarget = OUL;
278         link.SizeAsTarget = OUL;
279     }
280 }
281 }

```

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

```

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

```

```

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

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

```

```

124 /// <para></para>
125 /// </param>
126 [MethodImpl(MethodImplOptions.AggressiveInlining)]
127 protected override void SetPointers(IResizableDirectMemory memory)
128 {
129     _header = (LinksHeader<ulong>*)memory.Pointer;
130     _links = (RawLink<ulong>*)memory.Pointer;
131     SourcesTreeMethods = _createSourceTreeMethods();
132     TargetsTreeMethods = _createTargetTreeMethods();
133     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
134 }
135
136 /// <summary>
137 /// <para>
138 /// Resets the pointers.
139 /// </para>
140 /// <para></para>
141 /// </summary>
142 [MethodImpl(MethodImplOptions.AggressiveInlining)]
143 protected override void ResetPointers()
144 {
145     base.ResetPointers();
146     _links = null;
147     _header = null;
148 }
149
150 /// <summary>
151 /// <para>
152 /// Gets the header reference.
153 /// </para>
154 /// <para></para>
155 /// </summary>
156 /// <returns>
157 /// <para>A ref links header of ulong</para>
158 /// <para></para>
159 /// </returns>
160 [MethodImpl(MethodImplOptions.AggressiveInlining)]
161 protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
162
163 /// <summary>
164 /// <para>
165 /// Gets the link reference using the specified link index.
166 /// </para>
167 /// <para></para>
168 /// </summary>
169 /// <param name="linkIndex">
170 /// <para>The link index.</para>
171 /// <para></para>
172 /// </param>
173 /// <returns>
174 /// <para>A ref raw link of ulong</para>
175 /// <para></para>
176 /// </returns>
177 [MethodImpl(MethodImplOptions.AggressiveInlining)]
178 protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
    ↪ _links[linkIndex];
179
180 /// <summary>
181 /// <para>
182 /// Determines whether this instance are equal.
183 /// </para>
184 /// <para></para>
185 /// </summary>
186 /// <param name="first">
187 /// <para>The first.</para>
188 /// <para></para>
189 /// </param>
190 /// <param name="second">
191 /// <para>The second.</para>
192 /// <para></para>
193 /// </param>
194 /// <returns>
195 /// <para>The bool</para>
196 /// <para></para>
197 /// </returns>
198 [MethodImpl(MethodImplOptions.AggressiveInlining)]
199 protected override bool AreEqual(ulong first, ulong second) => first == second;
200

```

```

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

```

```

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

```

```

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

```

1.108 ./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  }

```

1.109 ./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,
    ↳ TLink>
18 {
19     private static readonly EqualityComparer<TLink> _equalityComparer =
        ↳ EqualityComparer<TLink>.Default;
20
21     /// <summary>
22     /// <para>
23     /// Initializes a new <see cref="PropertiesOperator"/> instance.
24     /// </para>
25     /// <para></para>
26     /// </summary>
27     /// <param name="links">
28     /// <para>A links.</para>
29     /// <para></para>
30     /// </param>
31     [MethodImpl(MethodImplOptions.AggressiveInlining)]
32     public PropertiesOperator(ILinks<TLink> links) : base(links) { }
33
34     /// <summary>
35     /// <para>
36     /// Gets the value using the specified object.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     /// <param name="@object">
41     /// <para>The object.</para>
42     /// <para></para>
43     /// </param>
44     /// <param name="property">
45     /// <para>The property.</para>
46     /// <para></para>
47     /// </param>
48     /// <returns>
49     /// <para>The link</para>
50     /// <para></para>
51     /// </returns>
52     [MethodImpl(MethodImplOptions.AggressiveInlining)]
53     public 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 }

```

1.110 ./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
22         private readonly TLink _propertyMarker;
23         private readonly TLink _propertyValueMarker;
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="PropertyOperator"/> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="links">
32         /// <para>A links.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="propertyMarker">
36         /// <para>A property marker.</para>
37         /// <para></para>
38         /// </param>
39         /// <param name="propertyValueMarker">
40         /// <para>A property value marker.</para>
41         /// <para></para>
42         /// </param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public PropertyOperator(ILinks<TLink> links, TLink propertyMarker, TLink
45             ↪ propertyValueMarker) : base(links)
46         {
47             _propertyMarker = propertyMarker;
48             _propertyValueMarker = propertyValueMarker;
49         }
50
51         /// <summary>
52         /// <para>
53         /// Gets the link.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         /// <param name="link">
58         /// <para>The link.</para>
59         /// <para></para>
60         /// </param>
61         /// <returns>
62         /// <para>The link</para>
63         /// <para></para>
64         /// </returns>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         public TLink Get(TLink link)
67         {
68             var property = _links.SearchOrDefault(link, _propertyMarker);
69             return GetValue(GetContainer(property));
70         }
71     }
72 }

```

```

69
70     /// <summary>
71     /// <para>
72     /// Gets the container using the specified property.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="property">
77     /// <para>The property.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The value container.</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     private TLink GetContainer(TLink property)
86     {
87         var valueContainer = default(TLink);
88         if (_equalityComparer.Equals(property, default))
89         {
90             return valueContainer;
91         }
92         var links = _links;
93         var constants = links.Constants;
94         var continueConstant = constants.Continue;
95         var breakConstant = constants.Break;
96         var anyConstant = constants.Any;
97         var query = new Link<TLink>(anyConstant, property, anyConstant);
98         links.Each(candidate =>
99         {
100             var candidateTarget = links.GetTarget(candidate);
101             var valueTarget = links.GetTarget(candidateTarget);
102             if (_equalityComparer.Equals(valueTarget, _propertyValueMarker))
103             {
104                 valueContainer = links.GetIndex(candidate);
105                 return breakConstant;
106             }
107             return continueConstant;
108         }, query);
109         return valueContainer;
110     }
111
112     /// <summary>
113     /// <para>
114     /// Gets the value using the specified container.
115     /// </para>
116     /// <para></para>
117     /// </summary>
118     /// <param name="container">
119     /// <para>The container.</para>
120     /// <para></para>
121     /// </param>
122     /// <returns>
123     /// <para>The link</para>
124     /// <para></para>
125     /// </returns>
126     [MethodImpl(MethodImplOptions.AggressiveInlining)]
127     private TLink GetValue(TLink container) => _equalityComparer.Equals(container, default)
128     ↪ ? default : _links.GetTarget(container);
129
130     /// <summary>
131     /// <para>
132     /// Sets the link.
133     /// </para>
134     /// <para></para>
135     /// </summary>
136     /// <param name="link">
137     /// <para>The link.</para>
138     /// <para></para>
139     /// </param>
140     /// <param name="value">
141     /// <para>The value.</para>
142     /// <para></para>
143     /// </param>
144     [MethodImpl(MethodImplOptions.AggressiveInlining)]
145     public void Set(TLink link, TLink value)
146     {

```

```

146     var links = _links;
147     var property = links.GetOrCreate(link, _propertyMarker);
148     var container = GetContainer(property);
149     if (_equalityComparer.Equals(container, default))
150     {
151         links.GetOrCreate(property, value);
152     }
153     else
154     {
155         links.Update(container, property, value);
156     }
157 }
158 }
159 }

```

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

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3 using Platform.Collections.Stacks;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Stacks
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the stack.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="LinksOperatorBase{TLink}">
16    /// <seealso cref="IStack{TLink}">
17    public class Stack<TLink> : LinksOperatorBase<TLink>, IStack<TLink>
18    {
19        private static readonly EqualityComparer<TLink> _equalityComparer =
20            ↪ EqualityComparer<TLink>.Default;
21
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></para>
40        /// </summary>
41        /// <param name="links">
42        /// <para>A links.</para>
43        /// <para></para>
44        /// </param>
45        /// <param name="stack">
46        /// <para>A stack.</para>
47        /// <para></para>
48        /// </param>
49        [MethodImpl(MethodImplOptions.AggressiveInlining)]
50        public Stack(ILinks<TLink> links, TLink stack) : base(links) => _stack = stack;
51
52        /// <summary>
53        /// <para>
54        /// Gets the stack marker.
55        /// </para>
56        /// <para></para>
57        /// </summary>
58        /// <returns>
59        /// <para>The link</para>
60        /// <para></para>
61        /// </returns>
62        [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

63     private TLink GetStackMarker() => _links.GetSource(_stack);
64
65     /// <summary>
66     /// <para>
67     /// Gets the top.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <returns>
72     /// <para>The link</para>
73     /// <para></para>
74     /// </returns>
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     private TLink GetTop() => _links.GetTarget(_stack);
77
78     /// <summary>
79     /// <para>
80     /// Peeks this instance.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <returns>
85     /// <para>The link</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public TLink Peek() => _links.GetTarget(GetTop());
90
91     /// <summary>
92     /// <para>
93     /// Pops this instance.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <returns>
98     /// <para>The element.</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    public TLink Pop()
103    {
104        var element = Peek();
105        if (!_equalityComparer.Equals(element, _stack))
106        {
107            var top = GetTop();
108            var previousTop = _links.GetSource(top);
109            _links.Update(_stack, GetStackMarker(), previousTop);
110            _links.Delete(top);
111        }
112        return element;
113    }
114
115    /// <summary>
116    /// <para>
117    /// Pushes the element.
118    /// </para>
119    /// <para></para>
120    /// </summary>
121    /// <param name="element">
122    /// <para>The element.</para>
123    /// <para></para>
124    /// </param>
125    [MethodImpl(MethodImplOptions.AggressiveInlining)]
126    public void Push(TLink element) => _links.Update(_stack, GetStackMarker(),
        ↪ _links.GetOrCreate(GetTop(), element));
127 }
128 }

```

1.112 ./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 }

```

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

```

```

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

```

```

118     /// <summary>
119     /// <para>
120     /// Eaches the handler.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="handler">
125     /// <para>The handler.</para>
126     /// <para></para>
127     /// </param>
128     /// <param name="restrictions">
129     /// <para>The restrictions.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The link address</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     public TLinkAddress Each(Func<IList<TLinkAddress>, TLinkAddress> handler,
138         ↪ IList<TLinkAddress> restrictions) => SyncRoot.ExecuteReadOperation(handler,
139         ↪ restrictions, (handler1, restrictions1) => Unsync.Each(handler1, restrictions1));
140
141     /// <summary>
142     /// <para>
143     /// Creates the restrictions.
144     /// </para>
145     /// <para></para>
146     /// </summary>
147     /// <param name="restrictions">
148     /// <para>The restrictions.</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(IList<TLinkAddress> restrictions) =>
157         ↪ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Create);
158
159     /// <summary>
160     /// <para>
161     /// Updates the restrictions.
162     /// </para>
163     /// <para></para>
164     /// </summary>
165     /// <param name="restrictions">
166     /// <para>The restrictions.</para>
167     /// <para></para>
168     /// </param>
169     /// <param name="substitution">
170     /// <para>The substitution.</para>
171     /// <para></para>
172     /// </param>
173     /// <returns>
174     /// <para>The link address</para>
175     /// <para></para>
176     /// </returns>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     public TLinkAddress Update(IList<TLinkAddress> restrictions, IList<TLinkAddress>
179         ↪ substitution) => SyncRoot.ExecuteWriteOperation(restrictions, substitution,
180         ↪ Unsync.Update);
181
182     /// <summary>
183     /// <para>
184     /// Deletes the restrictions.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="restrictions">
189     /// <para>The restrictions.</para>
190     /// <para></para>
191     /// </param>
192     [MethodImpl(MethodImplOptions.AggressiveInlining)]
193     public void Delete(IList<TLinkAddress> restrictions) =>
194         ↪ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Delete);

```

```

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

```

1.114 ./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 }

```



```

63     /// <summary>
64     /// <para>
65     /// Formats the structure using the specified links.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="links">
70     /// <para>The links.</para>
71     /// <para></para>
72     /// </param>
73     /// <param name="linkIndex">
74     /// <para>The link index.</para>
75     /// <para></para>
76     /// </param>
77     /// <param name="isElement">
78     /// <para>The is element.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="renderIndex">
82     /// <para>The render index.</para>
83     /// <para></para>
84     /// </param>
85     /// <param name="renderDebug">
86     /// <para>The render debug.</para>
87     /// <para></para>
88     /// </param>
89     /// <returns>
90     /// <para>The string</para>
91     /// <para></para>
92     /// </returns>
93     [MethodImpl(MethodImplOptions.AggressiveInlining)]
94     public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
95     ↪ Func<Link<ulong>, bool> isElement, bool renderIndex = false, bool renderDebug =
96     ↪ false)
97     {
98         var sb = new StringBuilder();
99         var visited = new HashSet<ulong>();
100         links.AppendStructure(sb, visited, linkIndex, isElement, (innerSb, link) =>
101         ↪ innerSb.Append(link.Index), renderIndex, renderDebug);
102         return sb.ToString();
103     }
104     /// <summary>
105     /// <para>
106     /// Formats the structure using the specified links.
107     /// </para>
108     /// <para></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)]

```

```

137 public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
    ↳ Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
    ↳ bool renderIndex = false, bool renderDebug = false)
138 {
139     var sb = new StringBuilder();
140     var visited = new HashSet<ulong>();
141     links.AppendStructure(sb, visited, linkIndex, isElement, appendElement, renderIndex,
    ↳ renderDebug);
142     return sb.ToString();
143 }
144
145 /// <summary>
146 /// <para>
147 /// Appends the structure using the specified links.
148 /// </para>
149 /// <para></para>
150 /// </summary>
151 /// <param name="links">
152 /// <para>The links.</para>
153 /// <para></para>
154 /// </param>
155 /// <param name="sb">
156 /// <para>The sb.</para>
157 /// <para></para>
158 /// </param>
159 /// <param name="visited">
160 /// <para>The visited.</para>
161 /// <para></para>
162 /// </param>
163 /// <param name="linkIndex">
164 /// <para>The link index.</para>
165 /// <para></para>
166 /// </param>
167 /// <param name="isElement">
168 /// <para>The is element.</para>
169 /// <para></para>
170 /// </param>
171 /// <param name="appendElement">
172 /// <para>The append element.</para>
173 /// <para></para>
174 /// </param>
175 /// <param name="renderIndex">
176 /// <para>The render index.</para>
177 /// <para></para>
178 /// </param>
179 /// <param name="renderDebug">
180 /// <para>The render debug.</para>
181 /// <para></para>
182 /// </param>
183 /// <exception cref="ArgumentNullException">
184 /// <para></para>
185 /// <para></para>
186 /// </exception>
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 public static void AppendStructure(this ILinks<ulong> links, StringBuilder sb,
    ↳ HashSet<ulong> visited, ulong linkIndex, Func<Link<ulong>, bool> isElement,
    ↳ Action<StringBuilder, Link<ulong>> appendElement, bool renderIndex = false, bool
    ↳ renderDebug = false)
189 {
190     if (sb == null)
191     {
192         throw new ArgumentNullException(nameof(sb));
193     }
194     if (linkIndex == Constants.Null || linkIndex == Constants.Any || linkIndex ==
    ↳ Constants.Itself)
195     {
196         return;
197     }
198     if (links.Exists(linkIndex))
199     {
200         if (visited.Add(linkIndex))
201         {
202             sb.Append('(');
203             var link = new Link<ulong>(links.GetLink(linkIndex));
204             if (renderIndex)
205             {
206                 sb.Append(link.Index);

```

```

207         sb.Append(':');
208     }
209     if (link.Source == link.Index)
210     {
211         sb.Append(link.Index);
212     }
213     else
214     {
215         var source = new Link<ulong>(links.GetLink(link.Source));
216         if (isElement(source))
217         {
218             appendElement(sb, source);
219         }
220         else
221         {
222             links.AppendStructure(sb, visited, source.Index, isElement,
223                 ↪ appendElement, renderIndex);
224         }
225     }
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 else
256 {
257     if (renderDebug)
258     {
259         sb.Append('~');
260     }
261     sb.Append(linkIndex);
262 }
263 }

```

1.115 ./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.Exceptions;
14
15 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
16
17 namespace Platform.Data.Doublets
18 {

```

```

19  /// <summary>
20  /// <para>
21  /// Represents the int 64 links transactions layer.
22  /// </para>
23  /// <para></para>
24  /// </summary>
25  /// <seealso cref="LinksDisposableDecoratorBase{ulong}" />
26  public class UInt64LinksTransactionsLayer : LinksDisposableDecoratorBase<ulong> //-V3073
27  {
28      /// <remarks>
29      /// Альтернативные варианты хранения трансформации (элемента транзакции):
30      ///
31      /// private enum TransitionType
32      /// {
33      ///     Creation,
34      ///     UpdateOf,
35      ///     UpdateTo,
36      ///     Deletion
37      /// }
38      ///
39      /// private struct Transition
40      /// {
41      ///     public ulong TransactionId;
42      ///     public UniqueTimestamp Timestamp;
43      ///     public TransactionItemType Type;
44      ///     public Link Source;
45      ///     public Link Linker;
46      ///     public Link Target;
47      /// }
48      ///
49      /// Или
50      ///
51      /// public struct TransitionHeader
52      /// {
53      ///     public ulong TransactionIdCombined;
54      ///     public ulong TimestampCombined;
55      ///
56      ///     public ulong TransactionId
57      ///     {
58      ///         get
59      ///         {
60      ///             return (ulong) mask & TransactionIdCombined;
61      ///         }
62      ///     }
63      ///
64      ///     public UniqueTimestamp Timestamp
65      ///     {
66      ///         get
67      ///         {
68      ///             return (UniqueTimestamp)mask & TransactionIdCombined;
69      ///         }
70      ///     }
71      ///
72      ///     public TransactionItemType Type
73      ///     {
74      ///         get
75      ///         {
76      ///             // Использовать по одному биту из TransactionId и Timestamp,
77      ///             // для значения в 2 бита, которое представляет тип операции
78      ///             throw new NotImplementedException();
79      ///         }
80      ///     }
81      /// }
82      ///
83      /// private struct Transition
84      /// {
85      ///     public TransitionHeader Header;
86      ///     public Link Source;
87      ///     public Link Linker;
88      ///     public Link Target;
89      /// }
90      ///
91      /// </remarks>
92  public struct Transition : IEquatable<Transition>
93  {
94      /// <summary>
95      /// <para>
96      /// The size.

```

```

97     /// </para>
98     /// <para></para>
99     /// </summary>
100     public static readonly long Size = Structure<Transition>.Size;
101
102     /// <summary>
103     /// <para>
104     /// The transaction id.
105     /// </para>
106     /// <para></para>
107     /// </summary>
108     public readonly ulong TransactionId;
109     /// <summary>
110     /// <para>
111     /// The before.
112     /// </para>
113     /// <para></para>
114     /// </summary>
115     public readonly Link<ulong> Before;
116     /// <summary>
117     /// <para>
118     /// The after.
119     /// </para>
120     /// <para></para>
121     /// </summary>
122     public readonly Link<ulong> After;
123     /// <summary>
124     /// <para>
125     /// The timestamp.
126     /// </para>
127     /// <para></para>
128     /// </summary>
129     public readonly Timestamp Timestamp;
130
131     /// <summary>
132     /// <para>
133     /// Initializes a new <see cref="Transition"/> instance.
134     /// </para>
135     /// <para></para>
136     /// </summary>
137     /// <param name="uniqueTimestampFactory">
138     /// <para>A unique timestamp factory.</para>
139     /// <para></para>
140     /// </param>
141     /// <param name="transactionId">
142     /// <para>A transaction id.</para>
143     /// <para></para>
144     /// </param>
145     /// <param name="before">
146     /// <para>A before.</para>
147     /// <para></para>
148     /// </param>
149     /// <param name="after">
150     /// <para>A after.</para>
151     /// <para></para>
152     /// </param>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId, Link<ulong> before, Link<ulong> after)
155     {
156         TransactionId = transactionId;
157         Before = before;
158         After = after;
159         Timestamp = uniqueTimestampFactory.Create();
160     }
161
162     /// <summary>
163     /// <para>
164     /// Initializes a new <see cref="Transition"/> instance.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="uniqueTimestampFactory">
169     /// <para>A unique timestamp factory.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="transactionId">
173     /// <para>A transaction id.</para>

```

```

174     /// <para></para>
175     /// </param>
176     /// <param name="before">
177     /// <para>A before.</para>
178     /// <para></para>
179     /// </param>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId, Link<ulong> before) : this(uniqueTimestampFactory, transactionId,
        ↳ before, default) { }

182
183     /// <summary>
184     /// <para>
185     /// Initializes a new <see cref="Transition"/> instance.
186     /// </para>
187     /// <para></para>
188     /// </summary>
189     /// <param name="uniqueTimestampFactory">
190     /// <para>A unique timestamp factory.</para>
191     /// <para></para>
192     /// </param>
193     /// <param name="transactionId">
194     /// <para>A transaction id.</para>
195     /// <para></para>
196     /// </param>
197     [MethodImpl(MethodImplOptions.AggressiveInlining)]
198     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
        ↳ }

199
200     /// <summary>
201     /// <para>
202     /// Returns the string.
203     /// </para>
204     /// <para></para>
205     /// </summary>
206     /// <returns>
207     /// <para>The string</para>
208     /// <para></para>
209     /// </returns>
210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
211     public override string ToString() => $"{Timestamp} {TransactionId}: {Before} =>
        ↳ {After}";

212
213     /// <summary>
214     /// <para>
215     /// Determines whether this instance equals.
216     /// </para>
217     /// <para></para>
218     /// </summary>
219     /// <param name="obj">
220     /// <para>The obj.</para>
221     /// <para></para>
222     /// </param>
223     /// <returns>
224     /// <para>The bool</para>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     public override bool Equals(object obj) => obj is Transition transition ?
        ↳ Equals(transition) : false;

229
230     /// <summary>
231     /// <para>
232     /// Gets the hash code.
233     /// </para>
234     /// <para></para>
235     /// </summary>
236     /// <returns>
237     /// <para>The int</para>
238     /// <para></para>
239     /// </returns>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     public override int GetHashCode() => (TransactionId, Before, After,
        ↳ Timestamp).GetHashCode();

242
243     /// <summary>

```

```

244     /// <para>
245     /// Determines whether this instance equals.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     /// <param name="other">
250     /// <para>The other.</para>
251     /// <para></para>
252     /// </param>
253     /// <returns>
254     /// <para>The bool</para>
255     /// <para></para>
256     /// </returns>
257     [MethodImpl(MethodImplOptions.AggressiveInlining)]
258     public bool Equals(Transition other) => TransactionId == other.TransactionId &&
        ↳ Before == other.Before && After == other.After && Timestamp == other.Timestamp;
259
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     public static bool operator ==(Transition left, Transition right) =>
        ↳ left.Equals(right);
262
263     [MethodImpl(MethodImplOptions.AggressiveInlining)]
264     public static bool operator !=(Transition left, Transition right) => !(left ==
        ↳ right);
265 }
266
267     /// <remarks>
268     /// Другие варианты реализации транзакций (атомарности):
269     ///     1. Разделение хранения значения связи ((Source Target) или (Source Linker
270     ↳ Target)) и индексов.
271     ///     2. Хранение трансформаций/операций в отдельном хранилище Links, но дополнительно
272     ↳ потребуется решить вопрос
273     ///         со ссылками на внешние идентификаторы, или как-то иначе решить вопрос с
274     ↳ 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
286     ↳ константой.
287     ///     3. При подтверждении транзакции (Commit) все трансформации записываются разом
288     ↳ создавая задержку.
289     ///
290     /// На жёстком диске:
291     /// Минусы:
292     ///     1. Длительный отклик, на запись каждой трансформации.
293     ///     2. Лог транзакций дополнительно наполняется отменёнными транзакциями.
294     ///     -> Это может решаться упаковкой/исключением дублирующих операций.
295     ///     -> Также это может решаться тем, что короткие транзакции вообще
296     ///     не будут записываться в случае отката.
297     ///     3. Перед тем как выполнять отмену операций транзакции нужно дождаться пока все
298     ↳ операции (трансформации)
299     ///     будут записаны в лог.
300     ///
301     /// </remarks>
302     public class Transaction : DisposableBase
303     {
304         private readonly Queue<Transition> _transitions;
305         private readonly UInt64LinksTransactionsLayer _layer;
306         /// <summary>
307         /// <para>
308         /// Gets or sets the is committed value.
309         /// </para>
310         /// <para></para>
311         /// </summary>
312         public bool IsCommitted { get; private set; }
313         /// <summary>
314         /// <para>
315         /// Gets or sets the is reverted value.
316         /// </para>
317         /// <para></para>
318         /// </summary>

```

```

313 public bool IsReverted { get; private set; }
314
315 /// <summary>
316 /// <para>
317 /// Initializes a new <see cref="Transaction"/> instance.
318 /// </para>
319 /// <para></para>
320 /// </summary>
321 /// <param name="layer">
322 /// <para>A layer.</para>
323 /// <para></para>
324 /// </param>
325 /// <exception cref="NotSupportedException">
326 /// <para>Nested transactions not supported.</para>
327 /// <para></para>
328 /// </exception>
329 [MethodImpl(MethodImplOptions.AggressiveInlining)]
330 public Transaction(UInt64LinksTransactionsLayer layer)
331 {
332     _layer = layer;
333     if (_layer._currentTransactionId != 0)
334     {
335         throw new NotSupportedException("Nested transactions not supported.");
336     }
337     IsCommitted = false;
338     IsReverted = false;
339     _transitions = new Queue<Transition>();
340     SetCurrentTransaction(layer, this);
341 }
342
343 /// <summary>
344 /// <para>
345 /// Commits this instance.
346 /// </para>
347 /// <para></para>
348 /// </summary>
349 [MethodImpl(MethodImplOptions.AggressiveInlining)]
350 public void Commit()
351 {
352     EnsureTransactionAllowsWriteOperations(this);
353     while (_transitions.Count > 0)
354     {
355         var transition = _transitions.Dequeue();
356         _layer._transitions.Enqueue(transition);
357     }
358     _layer._lastCommittedTransactionId = _layer._currentTransactionId;
359     IsCommitted = true;
360 }
361
362 /// <summary>
363 /// <para>
364 /// Reverts this instance.
365 /// </para>
366 /// <para></para>
367 /// </summary>
368 [MethodImpl(MethodImplOptions.AggressiveInlining)]
369 private void Revert()
370 {
371     EnsureTransactionAllowsWriteOperations(this);
372     var transitionsToRevert = new Transition[_transitions.Count];
373     _transitions.CopyTo(transitionsToRevert, 0);
374     for (var i = transitionsToRevert.Length - 1; i >= 0; i--)
375     {
376         _layer.RevertTransition(transitionsToRevert[i]);
377     }
378     IsReverted = true;
379 }
380
381 /// <summary>
382 /// <para>
383 /// Sets the current transaction using the specified layer.
384 /// </para>
385 /// <para></para>
386 /// </summary>
387 /// <param name="layer">
388 /// <para>The layer.</para>
389 /// <para></para>
390 /// </param>

```



```

391     /// <param name="transaction">
392     /// <para>The transaction.</para>
393     /// <para></para>
394     /// </param>
395     [MethodImpl(MethodImplOptions.AggressiveInlining)]
396     public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
397     ↪ Transaction transaction)
398     {
399         layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
400         layer._currentTransactionTransitions = transaction._transitions;
401         layer._currentTransaction = transaction;
402     }
403     /// <summary>
404     /// <para>
405     /// Ensures the transaction allows write operations using the specified transaction.
406     /// </para>
407     /// <para></para>
408     /// </summary>
409     /// <param name="transaction">
410     /// <para>The transaction.</para>
411     /// <para></para>
412     /// </param>
413     /// <exception cref="InvalidOperationException">
414     /// <para>Transation is committed.</para>
415     /// <para></para>
416     /// </exception>
417     /// <exception cref="InvalidOperationException">
418     /// <para>Transation is reverted.</para>
419     /// <para></para>
420     /// </exception>
421     [MethodImpl(MethodImplOptions.AggressiveInlining)]
422     public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
423     {
424         if (transaction.IsReverted)
425         {
426             throw new InvalidOperationException("Transation is reverted.");
427         }
428         if (transaction.IsCommitted)
429         {
430             throw new InvalidOperationException("Transation is committed.");
431         }
432     }
433     /// <summary>
434     /// <para>
435     /// Disposes the manual.
436     /// </para>
437     /// <para></para>
438     /// </summary>
439     /// <param name="manual">
440     /// <para>The manual.</para>
441     /// <para></para>
442     /// </param>
443     /// <param name="wasDisposed">
444     /// <para>The was disposed.</para>
445     /// <para></para>
446     /// </param>
447     [MethodImpl(MethodImplOptions.AggressiveInlining)]
448     protected override void Dispose(bool manual, bool wasDisposed)
449     {
450         if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
451         {
452             if (!IsCommitted && !IsReverted)
453             {
454                 Revert();
455             }
456             _layer.ResetCurrentTransation();
457         }
458     }
459 }
460
461 /// <summary>
462 /// <para>
463 /// The from seconds.
464 /// </para>
465 /// <para></para>
466 /// </summary>
467

```

```

468 public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
469
470 private readonly string _logAddress;
471 private readonly FileStream _log;
472 private readonly Queue<Transition> _transitions;
473 private readonly UniqueTimestampFactory _uniqueTimestampFactory;
474 private Task _transitionsPusher;
475 private Transition _lastCommittedTransition;
476 private ulong _currentTransactionId;
477 private Queue<Transition> _currentTransactionTransitions;
478 private Transaction _currentTransaction;
479 private ulong _lastCommittedTransactionId;
480
481 /// <summary>
482 /// <para>
483 /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.
484 /// </para>
485 /// <para></para>
486 /// </summary>
487 /// <param name="links">
488 /// <para>A links.</para>
489 /// <para></para>
490 /// </param>
491 /// <param name="logAddress">
492 /// <para>A log address.</para>
493 /// <para></para>
494 /// </param>
495 /// <exception cref="ArgumentNullException">
496 /// <para></para>
497 /// <para></para>
498 /// </exception>
499 /// <exception cref="NotSupportedException">
500 /// <para>Database is damaged, autorecovery is not supported yet.</para>
501 /// <para></para>
502 /// </exception>
503 [MethodImpl(MethodImplOptions.AggressiveInlining)]
504 public UInt64LinksTransactionsLayer(ILinks<ulong> links, string logAddress)
505     : base(links)
506 {
507     if (string.IsNullOrEmpty(logAddress))
508     {
509         throw new ArgumentNullException(nameof(logAddress));
510     }
511     // В первой строке файла хранится последняя закоммиченную транзакцию.
512     // При запуске это используется для проверки удачного закрытия файла лога.
513     // In the first line of the file the last committed transaction is stored.
514     // On startup, this is used to check that the log file is successfully closed.
515     var lastCommittedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
516     var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
517     if (!lastCommittedTransition.Equals(lastWrittenTransition))
518     {
519         Dispose();
520         throw new NotSupportedException("Database is damaged, autorecovery is not
521             ↳ supported yet.");
522     }
523     if (lastCommittedTransition == default)
524     {
525         FileHelpers.WriteFirst(logAddress, lastCommittedTransition);
526     }
527     _lastCommittedTransition = lastCommittedTransition;
528     // TODO: Think about a better way to calculate or store this value
529     var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
530     _lastCommittedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
531         ↳ x.TransactionId) : 0;
532     _uniqueTimestampFactory = new UniqueTimestampFactory();
533     _logAddress = logAddress;
534     _log = FileHelpers.Append(logAddress);
535     _transitions = new Queue<Transition>();
536     _transitionsPusher = new Task(TransitionsPusher);
537     _transitionsPusher.Start();
538 }
539
540 /// <summary>
541 /// <para>
542 /// Gets the link value using the specified link.
543 /// </para>
544 /// <para></para>
545 /// </summary>
546 /// <param name="link">

```

```

545 /// <para>The link.</para>
546 /// <para></para>
547 /// </param>
548 /// <returns>
549 /// <para>A list of ulong</para>
550 /// <para></para>
551 /// </returns>
552 [MethodImpl(MethodImplOptions.AggressiveInlining)]
553 public IList<ulong> GetLinkValue(ulong link) => _links.GetLink(link);
554
555 /// <summary>
556 /// <para>
557 /// Creates the restrictions.
558 /// </para>
559 /// <para></para>
560 /// </summary>
561 /// <param name="restrictions">
562 /// <para>The restrictions.</para>
563 /// <para></para>
564 /// </param>
565 /// <returns>
566 /// <para>The created link index.</para>
567 /// <para></para>
568 /// </returns>
569 [MethodImpl(MethodImplOptions.AggressiveInlining)]
570 public override ulong Create(IList<ulong> restrictions)
571 {
572     var createdLinkIndex = _links.Create();
573     var createdLink = new Link<ulong>(_links.GetLink(createdLinkIndex));
574     CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
575         ↪ default, createdLink));
576     return createdLinkIndex;
577 }
578
579 /// <summary>
580 /// <para>
581 /// Updates the restrictions.
582 /// </para>
583 /// <para></para>
584 /// </summary>
585 /// <param name="restrictions">
586 /// <para>The restrictions.</para>
587 /// <para></para>
588 /// </param>
589 /// <param name="substitution">
590 /// <para>The substitution.</para>
591 /// <para></para>
592 /// </param>
593 /// <returns>
594 /// <para>The link index.</para>
595 /// <para></para>
596 /// </returns>
597 [MethodImpl(MethodImplOptions.AggressiveInlining)]
598 public override ulong Update(IList<ulong> restrictions, IList<ulong> substitution)
599 {
600     var linkIndex = restrictions[_constants.IndexPart];
601     var beforeLink = new Link<ulong>(_links.GetLink(linkIndex));
602     linkIndex = _links.Update(restrictions, substitution);
603     var afterLink = new Link<ulong>(_links.GetLink(linkIndex));
604     CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
605         ↪ beforeLink, afterLink));
606     return linkIndex;
607 }
608
609 /// <summary>
610 /// <para>
611 /// Deletes the restrictions.
612 /// </para>
613 /// <para></para>
614 /// </summary>
615 /// <param name="restrictions">
616 /// <para>The restrictions.</para>
617 /// <para></para>
618 /// </param>
619 [MethodImpl(MethodImplOptions.AggressiveInlining)]
620 public override void Delete(IList<ulong> restrictions)
621 {
622     var link = restrictions[_constants.IndexPart];

```

```

621     var deletedLink = new Link<ulong>(_links.GetLink(link));
622     _links.Delete(link);
623     CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
        ↪ deletedLink, default));
624 }
625
626 /// <summary>
627 /// <para>
628 /// Gets the current transitions.
629 /// </para>
630 /// <para></para>
631 /// </summary>
632 /// <returns>
633 /// <para>A queue of transition</para>
634 /// <para></para>
635 /// </returns>
636 [MethodImpl(MethodImplOptions.AggressiveInlining)]
637 private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
    ↪ _transitions;
638
639 /// <summary>
640 /// <para>
641 /// Commits the transition using the specified transition.
642 /// </para>
643 /// <para></para>
644 /// </summary>
645 /// <param name="transition">
646 /// <para>The transition.</para>
647 /// <para></para>
648 /// </param>
649 [MethodImpl(MethodImplOptions.AggressiveInlining)]
650 private void CommitTransition(Transition transition)
651 {
652     if (_currentTransaction != null)
653     {
654         Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
655     }
656     var transitions = GetCurrentTransitions();
657     transitions.Enqueue(transition);
658 }
659
660 /// <summary>
661 /// <para>
662 /// Reverts the transition using the specified transition.
663 /// </para>
664 /// <para></para>
665 /// </summary>
666 /// <param name="transition">
667 /// <para>The transition.</para>
668 /// <para></para>
669 /// </param>
670 [MethodImpl(MethodImplOptions.AggressiveInlining)]
671 private void RevertTransition(Transition transition)
672 {
673     if (transition.After.IsNull()) // Revert Deletion with Creation
674     {
675         _links.Create();
676     }
677     else if (transition.Before.IsNull()) // Revert Creation with Deletion
678     {
679         _links.Delete(transition.After.Index);
680     }
681     else // Revert Update
682     {
683         _links.Update(new[] { transition.After.Index, transition.Before.Source,
        ↪ transition.Before.Target });
684     }
685 }
686
687 /// <summary>
688 /// <para>
689 /// Resets the current transation.
690 /// </para>
691 /// <para></para>
692 /// </summary>
693 [MethodImpl(MethodImplOptions.AggressiveInlining)]
694 private void ResetCurrentTransation()
695 {

```

```

696     _currentTransactionId = 0;
697     _currentTransactionTransitions = null;
698     _currentTransaction = null;
699 }
700
701 /// <summary>
702 /// <para>
703 /// Pushes the transitions.
704 /// </para>
705 /// <para></para>
706 /// </summary>
707 [MethodImpl(MethodImplOptions.AggressiveInlining)]
708 private void PushTransitions()
709 {
710     if (_log == null || _transitions == null)
711     {
712         return;
713     }
714     for (var i = 0; i < _transitions.Count; i++)
715     {
716         var transition = _transitions.Dequeue();
717
718         _log.Write(transition);
719         _lastCommittedTransition = transition;
720     }
721 }
722
723 /// <summary>
724 /// <para>
725 /// Transitiones the pusher.
726 /// </para>
727 /// <para></para>
728 /// </summary>
729 [MethodImpl(MethodImplOptions.AggressiveInlining)]
730 private void TransitionsPusher()
731 {
732     while (!Disposable.IsDisposed && _transitionsPusher != null)
733     {
734         Thread.Sleep(DefaultPushDelay);
735         PushTransitions();
736     }
737 }
738
739 /// <summary>
740 /// <para>
741 /// Begins the transaction.
742 /// </para>
743 /// <para></para>
744 /// </summary>
745 /// <returns>
746 /// <para>The transaction</para>
747 /// <para></para>
748 /// </returns>
749 [MethodImpl(MethodImplOptions.AggressiveInlining)]
750 public Transaction BeginTransaction() => new Transaction(this);
751
752 /// <summary>
753 /// <para>
754 /// Disposes the transitions.
755 /// </para>
756 /// <para></para>
757 /// </summary>
758 [MethodImpl(MethodImplOptions.AggressiveInlining)]
759 private void DisposeTransitions()
760 {
761     try
762     {
763         var pusher = _transitionsPusher;
764         if (pusher != null)
765         {
766             _transitionsPusher = null;
767             pusher.Wait();
768         }
769         if (_transitions != null)
770         {
771             PushTransitions();
772         }
773         _log.DisposeIfPossible();
774         FileHelpers.WriteFirst(_logAddress, _lastCommittedTransition);

```

```

775     }
776     catch (Exception ex)
777     {
778         ex.Ignore();
779     }
780 }
781
782 #region DisposalBase
783
784 /// <summary>
785 /// <para>
786 /// Disposes the manual.
787 /// </para>
788 /// <para></para>
789 /// </summary>
790 /// <param name="manual">
791 /// <para>The manual.</para>
792 /// <para></para>
793 /// </param>
794 /// <param name="wasDisposed">
795 /// <para>The was disposed.</para>
796 /// <para></para>
797 /// </param>
798 [MethodImpl(MethodImplOptions.AggressiveInlining)]
799 protected override void Dispose(bool manual, bool wasDisposed)
800 {
801     if (!wasDisposed)
802     {
803         DisposeTransitions();
804     }
805     base.Dispose(manual, wasDisposed);
806 }
807
808 #endregion
809 }
810 }

```

1.116 ./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.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.Generic;
7
8  namespace Platform.Data.Doublets.Tests
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the generic links tests.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public unsafe static class GenericLinksTests
17     {
18         /// <summary>
19         /// <para>
20         /// Tests that crud test.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         [Fact]
25         public static void CRUDTest()
26         {
27             Using<byte>(links => links.TestCRUDOperations());
28             Using<ushort>(links => links.TestCRUDOperations());
29             Using<uint>(links => links.TestCRUDOperations());
30             Using<ulong>(links => links.TestCRUDOperations());
31         }
32
33         /// <summary>
34         /// <para>
35         /// Tests that raw numbers crud test.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         [Fact]
40         public static void RawNumbersCRUDTest()
41         {

```

```

42     Using<byte>(links => links.TestRowNumbersCRUDOperations());
43     Using<ushort>(links => links.TestRowNumbersCRUDOperations());
44     Using<uint>(links => links.TestRowNumbersCRUDOperations());
45     Using<ulong>(links => links.TestRowNumbersCRUDOperations());
46 }
47
48 /// <summary>
49 /// <para>
50 /// Tests that multiple random creations and deletions test.
51 /// </para>
52 /// <para></para>
53 /// </summary>
54 [Fact]
55 public static void MultipleRandomCreationsAndDeletionsTest()
56 {
57     Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
    ↪ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
    ↪ implementation of tree cuts out 5 bits from the address space.
58     Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
    ↪ stMultipleRandomCreationsAndDeletions(100));
59     Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
    ↪ MultipleRandomCreationsAndDeletions(100));
60     Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
    ↪ tMultipleRandomCreationsAndDeletions(100));
61 }
62
63 private static void Using<TLink>(Action<ILinks<TLink>> action)
64 {
65     using (var scope = new Scope<Types<HeapResizableDirectMemory,
    ↪ UnitedMemoryLinks<TLink>>>())
66     {
67         action(scope.Use<ILinks<TLink>>());
68     }
69 }
70 }
71 }

```

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

```

1  using Xunit;
2
3  namespace Platform.Data.Doublets.Tests
4  {
5      /// <summary>
6      /// <para>
7      /// Represents the links constants tests.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public static class LinksConstantsTests
12     {
13         /// <summary>
14         /// <para>
15         /// Tests that external references test.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         [Fact]
20         public static void ExternalReferencesTest()
21         {
22             LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue),
    ↪ (long.MaxValue + 1UL, ulong.MaxValue));
23
24             //var minimum = new Hybrid<ulong>(0, isExternal: true);
25             var minimum = new Hybrid<ulong>(1, isExternal: true);
26             var maximum = new Hybrid<ulong>(long.MaxValue, isExternal: true);
27
28             Assert.True(constants.IsExternalReference(minimum));
29             Assert.True(constants.IsExternalReference(maximum));
30         }
31     }
32 }

```

1.118 ./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;

```

```

7 namespace Platform.Data.Doublets.Tests
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the resizable direct memory links tests.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    public static class ResizableDirectMemoryLinksTests
16    {
17        private static readonly LinksConstants<ulong> _constants =
18            ↳ Default<LinksConstants<ulong>>.Instance;
19
20        /// <summary>
21        /// <para>
22        /// Tests that basic file mapped memory test.
23        /// </para>
24        /// <para></para>
25        /// </summary>
26        [Fact]
27        public static void BasicFileMappedMemoryTest()
28        {
29            var tempFilename = Path.GetTempFileName();
30            using (var memoryAdapter = new UInt64UnitedMemoryLinks(tempFilename))
31            {
32                memoryAdapter.TestBasicMemoryOperations();
33            }
34            File.Delete(tempFilename);
35
36            /// <summary>
37            /// <para>
38            /// Tests that basic heap memory test.
39            /// </para>
40            /// <para></para>
41            /// </summary>
42            [Fact]
43            public static void BasicHeapMemoryTest()
44            {
45                using (var memory = new
46                    ↳ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
47                using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
48                    ↳ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
49                {
50                    memoryAdapter.TestBasicMemoryOperations();
51                }
52
53                private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
54                {
55                    var link = memoryAdapter.Create();
56                    memoryAdapter.Delete(link);
57                }
58
59                /// <summary>
60                /// <para>
61                /// Tests that nonexistent references heap memory test.
62                /// </para>
63                /// <para></para>
64                /// </summary>
65                [Fact]
66                public static void NonexistentReferencesHeapMemoryTest()
67                {
68                    using (var memory = new
69                        ↳ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
70                    using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
71                        ↳ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
72                    {
73                        memoryAdapter.TestNonexistentReferences();
74                    }
75
76                private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
77                {
78                    var link = memoryAdapter.Create();
79                    memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);
80                    var resultLink = constants.Null;
81                }
82            }
83        }
84    }
85 }

```



```

79     memoryAdapter.Each(foundLink =>
80     {
81         resultLink = foundLink[_constants.IndexPart];
82         return _constants.Break;
83     }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
84     Assert.True(resultLink == link);
85     Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
86     memoryAdapter.Delete(link);
87 }
88 }
89 }

```

1.119 ./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     /// <summary>
12     /// <para>
13     /// Represents the scope tests.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class ScopeTests
18     {
19         /// <summary>
20         /// <para>
21         /// Tests that single dependency test.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         [Fact]
26         public static void SingleDependencyTest()
27         {
28             using (var scope = new Scope())
29             {
30                 scope.IncludeAssemblyOf<IMemory>();
31                 var instance = scope.Use<IDirectMemory>();
32                 Assert.IsType<HeapResizableDirectMemory>(instance);
33             }
34         }
35
36         /// <summary>
37         /// <para>
38         /// Tests that cascade dependency test.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         [Fact]
43         public static void CascadeDependencyTest()
44         {
45             using (var scope = new Scope())
46             {
47                 scope.Include<TemporaryFileMappedResizableDirectMemory>();
48                 scope.Include<UInt64UnitedMemoryLinks>();
49                 var instance = scope.Use<ILinks<ulong>>();
50                 Assert.IsType<UInt64UnitedMemoryLinks>(instance);
51             }
52         }
53
54         /// <summary>
55         /// <para>
56         /// Tests that full auto resolution test.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         [Fact(Skip = "Would be fixed later.")]
61         public static void FullAutoResolutionTest()
62         {
63             using (var scope = new Scope(autoInclude: true, autoExplore: true))
64             {
65                 var instance = scope.Use<UInt64Links>();
66                 Assert.IsType<UInt64Links>(instance);

```

```

67     }
68 }
69
70 /// <summary>
71 /// <para>
72 /// Tests that type parameters test.
73 /// </para>
74 /// <para></para>
75 /// </summary>
76 [Fact]
77 public static void TypeParametersTest()
78 {
79     using (var scope = new Scope<Types<HeapResizableDirectMemory,
80 ↪     UnitedMemoryLinks<ulong>>>())
81     {
82         var links = scope.Use<ILinks<ulong>>>();
83         Assert.IsType<UnitedMemoryLinks<ulong>>>(links);
84     }
85 }
86 }

```

1.120 ./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     /// <summary>
10    /// <para>
11    /// Represents the split memory generic links tests.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    public unsafe static class SplitMemoryGenericLinksTests
16    {
17        /// <summary>
18        /// <para>
19        /// Tests that crud test.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        [Fact]
24        public static void CRUDTest()
25        {
26            Using<byte>(links => links.TestCRUDOperations());
27            Using<ushort>(links => links.TestCRUDOperations());
28            Using<uint>(links => links.TestCRUDOperations());
29            Using<ulong>(links => links.TestCRUDOperations());
30        }
31
32        /// <summary>
33        /// <para>
34        /// Tests that raw numbers crud test.
35        /// </para>
36        /// <para></para>
37        /// </summary>
38        [Fact]
39        public static void RawNumbersCRUDTest()
40        {
41            UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
42            UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
43            UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
44            UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
45        }
46
47        /// <summary>
48        /// <para>
49        /// Tests that multiple random creations and deletions test.
50        /// </para>
51        /// <para></para>
52        /// </summary>
53        [Fact]
54        public static void MultipleRandomCreationsAndDeletionsTest()
55        {

```

```

56     Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
    ↪ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
    ↪ implementation of tree cuts out 5 bits from the address space.
57     Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
    ↪ stMultipleRandomCreationsAndDeletions(100));
58     Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
    ↪ MultipleRandomCreationsAndDeletions(100));
59     Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
    ↪ tMultipleRandomCreationsAndDeletions(100));
60 }
61
62 private static void Using<TLink>(Action<ILinks<TLink>> action)
63 {
64     using (var dataMemory = new HeapResizableDirectMemory())
65     using (var indexMemory = new HeapResizableDirectMemory())
66     using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory))
67     {
68         action(memory);
69     }
70 }
71
72 private static void UsingWithExternalReferences<TLink>(Action<ILinks<TLink>> action)
73 {
74     var contants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
75     using (var dataMemory = new HeapResizableDirectMemory())
76     using (var indexMemory = new HeapResizableDirectMemory())
77     using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory,
    ↪ SplitMemoryLinks<TLink>.DefaultLinksSizeStep, contants))
78     {
79         action(memory);
80     }
81 }
82 }
83 }

```

1.121 ./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      /// <summary>
10     /// <para>
11     /// Represents the split memory int 32 links tests.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public unsafe static class SplitMemoryUInt32LinksTests
16     {
17         /// <summary>
18         /// <para>
19         /// Tests that crud test.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         [Fact]
24         public static void CRUDTest()
25         {
26             Using(links => links.TestCRUDOperations());
27         }
28
29         /// <summary>
30         /// <para>
31         /// Tests that raw numbers crud test.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         [Fact]
36         public static void RawNumbersCRUDTest()
37         {
38             UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
39         }
40
41         /// <summary>
42         /// <para>

```

```

43     /// Tests that multiple random creations and deletions test.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     [Fact]
48     public static void MultipleRandomCreationsAndDeletionsTest()
49     {
50         Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
            ↳ leRandomCreationsAndDeletions(500));
51     }
52
53     private static void Using(Action<ILinks<TLink>> action)
54     {
55         using (var dataMemory = new HeapResizableDirectMemory())
56         using (var indexMemory = new HeapResizableDirectMemory())
57         using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory))
58         {
59             action(memory);
60         }
61     }
62
63     private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
64     {
65         var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
66         using (var dataMemory = new HeapResizableDirectMemory())
67         using (var indexMemory = new HeapResizableDirectMemory())
68         using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory,
            ↳ UInt32SplitMemoryLinks.DefaultLinksSizeStep, constants))
69         {
70             action(memory);
71         }
72     }
73 }
74 }

```

1.122 ./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      /// <summary>
10     /// <para>
11     /// Represents the split memory int 64 links tests.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public unsafe static class SplitMemoryUInt64LinksTests
16     {
17         /// <summary>
18         /// <para>
19         /// Tests that crud test.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         [Fact]
24         public static void CRUDTest()
25         {
26             Using(links => links.TestCRUDOperations());
27         }
28
29         /// <summary>
30         /// <para>
31         /// Tests that raw numbers crud test.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         [Fact]
36         public static void RawNumbersCRUDTest()
37         {
38             UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
39         }
40
41         /// <summary>
42         /// <para>

```

```

43     /// Tests that multiple random creations and deletions test.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     [Fact]
48     public static void MultipleRandomCreationsAndDeletionsTest()
49     {
50         Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
            ↪ leRandomCreationsAndDeletions(500));
51     }
52
53     private static void Using(Action<ILinks<TLink>> action)
54     {
55         using (var dataMemory = new HeapResizableDirectMemory())
56         using (var indexMemory = new HeapResizableDirectMemory())
57         using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory))
58         {
59             action(memory);
60         }
61     }
62
63     private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
64     {
65         var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
66         using (var dataMemory = new HeapResizableDirectMemory())
67         using (var indexMemory = new HeapResizableDirectMemory())
68         using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
            ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, constants))
69         {
70             action(memory);
71         }
72     }
73 }
74 }

```

1.123 ./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     /// <summary>
12     /// <para>
13     /// Represents the test extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class TestExtensions
18     {
19         /// <summary>
20         /// <para>
21         /// Tests the crud operations using the specified links.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <typeparam name="T">
26         /// <para>The .</para>
27         /// <para></para>
28         /// </typeparam>
29         /// <param name="links">
30         /// <para>The links.</para>
31         /// <para></para>
32         /// </param>
33         public static void TestCRUDOperations<T>(this ILinks<T> links)
34         {
35             var constants = links.Constants;
36
37             var equalityComparer = EqualityComparer<T>.Default;
38
39             var zero = default(T);
40             var one = Arithmetic.Increment(zero);
41
42             // Create Link
43             Assert.True(equalityComparer.Equals(links.Count(), zero));

```

```

44
45     var setter = new Setter<T>(constants.Null);
46     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
47
48     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
49
50     var linkAddress = links.Create();
51
52     var link = new Link<T>(links.GetLink(linkAddress));
53
54     Assert.True(link.Count == 3);
55     Assert.True(equalityComparer.Equals(link.Index, linkAddress));
56     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
57     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
58
59     Assert.True(equalityComparer.Equals(links.Count(), one));
60
61     // Get first link
62     setter = new Setter<T>(constants.Null);
63     links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
64
65     Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
66
67     // Update link to reference itself
68     links.Update(linkAddress, linkAddress, linkAddress);
69
70     link = new Link<T>(links.GetLink(linkAddress));
71
72     Assert.True(equalityComparer.Equals(link.Source, linkAddress));
73     Assert.True(equalityComparer.Equals(link.Target, linkAddress));
74
75     // Update link to reference null (prepare for delete)
76     var updated = links.Update(linkAddress, constants.Null, constants.Null);
77
78     Assert.True(equalityComparer.Equals(updated, linkAddress));
79
80     link = new Link<T>(links.GetLink(linkAddress));
81
82     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
83     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
84
85     // Delete link
86     links.Delete(linkAddress);
87
88     Assert.True(equalityComparer.Equals(links.Count(), zero));
89
90     setter = new Setter<T>(constants.Null);
91     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
92
93     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
94 }
95
96 /// <summary>
97 /// <para>
98 /// Tests the raw numbers crud operations using the specified links.
99 /// </para>
100 /// <para></para>
101 /// </summary>
102 /// <typeparam name="T">
103 /// <para>The .</para>
104 /// <para></para>
105 /// </typeparam>
106 /// <param name="links">
107 /// <para>The links.</para>
108 /// <para></para>
109 /// </param>
110 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
111 {
112     // Constants
113     var constants = links.Constants;
114     var equalityComparer = EqualityComparer<T>.Default;
115
116     var zero = default(T);
117     var one = Arithmetic.Increment(zero);
118     var two = Arithmetic.Increment(one);
119
120     var h106E = new Hybrid<T>(106L, isExternal: true);
121     var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
122     var h108E = new Hybrid<T>(-108L);
123

```

```

124 Assert.Equal(106L, h106E.AbsoluteValue);
125 Assert.Equal(107L, h107E.AbsoluteValue);
126 Assert.Equal(108L, h108E.AbsoluteValue);
127
128 // Create Link (External -> External)
129 var linkAddress1 = links.Create();
130
131 links.Update(linkAddress1, h106E, h108E);
132
133 var link1 = new Link<T>(links.GetLink(linkAddress1));
134
135 Assert.True(equalityComparer.Equals(link1.Source, h106E));
136 Assert.True(equalityComparer.Equals(link1.Target, h108E));
137
138 // Create Link (Internal -> External)
139 var linkAddress2 = links.Create();
140
141 links.Update(linkAddress2, linkAddress1, h108E);
142
143 var link2 = new Link<T>(links.GetLink(linkAddress2));
144
145 Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
146 Assert.True(equalityComparer.Equals(link2.Target, h108E));
147
148 // Create Link (Internal -> Internal)
149 var linkAddress3 = links.Create();
150
151 links.Update(linkAddress3, linkAddress1, linkAddress2);
152
153 var link3 = new Link<T>(links.GetLink(linkAddress3));
154
155 Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
156 Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
157
158 // Search for created link
159 var setter1 = new Setter<T>(constants.Null);
160 links.Each(h106E, h108E, setter1.SetAndReturnFalse);
161
162 Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
163
164 // Search for nonexistent link
165 var setter2 = new Setter<T>(constants.Null);
166 links.Each(h106E, h107E, setter2.SetAndReturnFalse);
167
168 Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
169
170 // Update link to reference null (prepare for delete)
171 var updated = links.Update(linkAddress3, constants.Null, constants.Null);
172
173 Assert.True(equalityComparer.Equals(updated, linkAddress3));
174
175 link3 = new Link<T>(links.GetLink(linkAddress3));
176
177 Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
178 Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
179
180 // Delete link
181 links.Delete(linkAddress3);
182
183 Assert.True(equalityComparer.Equals(links.Count(), two));
184
185 var setter3 = new Setter<T>(constants.Null);
186 links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
187
188 Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
189 }
190
191 /// <summary>
192 /// <para>
193 /// Tests the multiple random creations and deletions using the specified links.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <typeparam name="TLink">
198 /// <para>The link.</para>
199 /// <para></para>
200 /// </typeparam>
201 /// <param name="links">
202 /// <para>The links.</para>
203 /// <para></para>

```

```

204     /// </param>
205     /// <param name="maximumOperationsPerCycle">
206     /// <para>The maximum operations per cycle.</para>
207     /// </para></para>
208     /// </param>
209     public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
    → links, int maximumOperationsPerCycle)
210     {
211         var comparer = Comparer<TLink>.Default;
212         var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
213         var uint64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
214         for (var N = 1; N < maximumOperationsPerCycle; N++)
215         {
216             var random = new System.Random(N);
217             var created = 0UL;
218             var deleted = 0UL;
219             for (var i = 0; i < N; i++)
220             {
221                 var linksCount = addressToUInt64Converter.Convert(links.Count());
222                 var createPoint = random.NextBoolean();
223                 if (linksCount >= 2 && createPoint)
224                 {
225                     var linksAddressRange = new Range<ulong>(1, linksCount);
226                     TLink source = uint64ToAddressConverter.Convert(random.NextUInt64(linksA
    → ddressRange));
227                     TLink target = uint64ToAddressConverter.Convert(random.NextUInt64(linksA
    → ddressRange));
228                     → //-V3086
229                     var resultLink = links.GetOrCreate(source, target);
230                     if (comparer.Compare(resultLink,
    → uint64ToAddressConverter.Convert(linksCount)) > 0)
231                     {
232                         created++;
233                     }
234                     else
235                     {
236                         links.Create();
237                         created++;
238                     }
239                 }
240                 Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
241                 for (var i = 0; i < N; i++)
242                 {
243                     TLink link = uint64ToAddressConverter.Convert((ulong)i + 1UL);
244                     if (links.Exists(link))
245                     {
246                         links.Delete(link);
247                         deleted++;
248                     }
249                 }
250                 Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
251             }
252         }
253     }
254 }

```

1.124 ./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     /// <summary>
13     /// <para>
14     /// Represents the uint 64 links extensions tests.
15     /// </para>
16     /// </para></para>
17     /// </summary>
18     public class UInt64LinksExtensionsTests
19     {
20         /// <summary>
21         /// <para>

```



```

22     /// Creates the links.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <returns>
27     /// <para>A links of t link</para>
28     /// <para></para>
29     /// </returns>
30     public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
        ↳ Platform.IO.TemporaryFile());
31
32     /// <summary>
33     /// <para>
34     /// Creates the links using the specified data db filename.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <typeparam name="TLink">
39     /// <para>The link.</para>
40     /// <para></para>
41     /// </typeparam>
42     /// <param name="dataDBFilename">
43     /// <para>The data db filename.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>A links of t link</para>
48     /// <para></para>
49     /// </returns>
50     public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
51     {
52         var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
            ↳ true);
53         return new UnitedMemoryLinks<TLink>(new
            ↳ FileMappedResizableDirectMemory(dataDBFilename),
            ↳ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
            ↳ IndexTreeType.Default);
54     }
55     /// <summary>
56     /// <para>
57     /// Tests that format structure with external reference test.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     [Fact]
62     public void FormatStructureWithExternalReferenceTest()
63     {
64         ILinks<TLink> links = CreateLinks();
65         TLink zero = default;
66         var one = Arithmetic.Increment(zero);
67         var markerIndex = one;
68         var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
69         var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
            ↳ markerIndex));
70         AddressToRawNumberConverter<TLink> addressToNumberConverter = new();
71         var numberAddress = addressToNumberConverter.Convert(1);
72         var numberLink = links.GetOrCreate(numberMarker, numberAddress);
73         var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
            ↳ true);
74         Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
75     }
76 }
77 }

```

1.125 ./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     /// <summary>
12     /// <para>
13     /// Represents the united memory int 32 links tests.

```

```

14    /// </para>
15    /// <para></para>
16    /// </summary>
17    public unsafe static class UnitedMemoryUInt32LinksTests
18    {
19        /// <summary>
20        /// <para>
21        /// Tests that crud test.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        [Fact]
26        public static void CRUDTest()
27        {
28            Using(links => links.TestCRUDOperations());
29        }
30
31        /// <summary>
32        /// <para>
33        /// Tests that raw numbers crud test.
34        /// </para>
35        /// <para></para>
36        /// </summary>
37        [Fact]
38        public static void RawNumbersCRUDTest()
39        {
40            Using(links => links.TestRawNumbersCRUDOperations());
41        }
42
43        /// <summary>
44        /// <para>
45        /// Tests that multiple random creations and deletions test.
46        /// </para>
47        /// <para></para>
48        /// </summary>
49        [Fact]
50        public static void MultipleRandomCreationsAndDeletionsTest()
51        {
52            Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
53        }
54
55        private static void Using(Action<ILinks<TLink>> action)
56        {
57            using (var scope = new Scope<Types<HeapResizableDirectMemory,
58                ↳ UInt32UnitedMemoryLinks>>())
59            {
60                action(scope.Use<ILinks<TLink>>());
61            }
62        }
63    }

```

1.126 ./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     /// <summary>
12     /// <para>
13     /// Represents the united memory int 64 links tests.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public unsafe static class UnitedMemoryUInt64LinksTests
18     {
19         /// <summary>
20         /// <para>
21         /// Tests that crud test.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         [Fact]

```

```

26 public static void CRUDTest()
27 {
28     Using(links => links.TestCRUDOperations());
29 }
30
31 /// <summary>
32 /// <para>
33 /// Tests that raw numbers crud test.
34 /// </para>
35 /// <para></para>
36 /// </summary>
37 [Fact]
38 public static void RawNumbersCRUDTest()
39 {
40     Using(links => links.TestRawNumbersCRUDOperations());
41 }
42
43 /// <summary>
44 /// <para>
45 /// Tests that multiple random creations and deletions test.
46 /// </para>
47 /// <para></para>
48 /// </summary>
49 [Fact]
50 public static void MultipleRandomCreationsAndDeletionsTest()
51 {
52     Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
53 }
54
55 private static void Using(Action<ILinks<TLink>> action)
56 {
57     using (var scope = new Scope<Types<HeapResizableDirectMemory,
58         ↳ UInt64UnitedMemoryLinks>>())
59     {
60         action(scope.Use<ILinks<TLink>>());
61     }
62 }
63 }

```

Index

./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.cs, 446
./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs, 447
./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs, 447
./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs, 449
./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs, 450
./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs, 451
./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs, 452
./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs, 453
./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs, 456
./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs, 457
./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt64LinksTests.cs, 458
./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, 6
./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/NonNullContentsLinkDeletionResolver.cs, 14
./csharp/Platform.Data.Doublets/Decorators/UInt32Links.cs, 14
./csharp/Platform.Data.Doublets/Decorators/UInt64Links.cs, 16
./csharp/Platform.Data.Doublets/Decorators/UniLinks.cs, 18
./csharp/Platform.Data.Doublets/Doublet.cs, 24
./csharp/Platform.Data.Doublets/DoubletComparer.cs, 26
./csharp/Platform.Data.Doublets/ILinks.cs, 27
./csharp/Platform.Data.Doublets/ILinksExtensions.cs, 27
./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs, 46
./csharp/Platform.Data.Doublets/Link.cs, 46
./csharp/Platform.Data.Doublets/LinkExtensions.cs, 54
./csharp/Platform.Data.Doublets/LinksOperatorBase.cs, 55
./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs, 55
./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs, 56
./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs, 57
./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs, 58
./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 60
./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs, 67
./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 74
./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs, 78
./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 82
./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs, 86
./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 89
./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs, 95
./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesLinkedListMethods.cs, 101
./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 106
./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs, 110
./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 113
./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs, 117
./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs, 121
./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs, 124
./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs, 142
./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs, 145
./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs, 146
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 148
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase.cs, 154
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 160
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs, 164
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 168
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs, 172
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 176

./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 181
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 187
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 188
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 192
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 195
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 199
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 203
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs, 209
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 210
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase.cs, 216
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 222
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods.cs, 226
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 230
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethods.cs, 234
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 238
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.cs, 243
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 249
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 250
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 254
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 257
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 261
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 264
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 271
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 272
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 281
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 288
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 294
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 300
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 303
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 307
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 312
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 316
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs, 320
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs, 322
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs, 335
./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs, 338
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 340
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 346
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 351
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs, 355
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 359
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 362
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 366
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 372
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 373
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 380
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 386
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 391
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 396
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 400
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 404
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 409
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 413
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 417
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs, 422
./csharp/Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs, 423
./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs, 425
./csharp/Platform.Data.Doublets/Stacks/Stack.cs, 427
./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs, 428
./csharp/Platform.Data.Doublets/SynchronizedLinks.cs, 429
./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs, 432
./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 435