

LinksPlatform's Platform.Data.Doublets Class Library

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

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

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

```
1 using System.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        /// <summary>
```

```

17     /// <para>
18     /// Initializes a new <see cref="LinksCascadeUniquenessAndUsagesResolver"/> instance.
19     /// </para>
20     /// </summary>
21     /// <param name="links">
22     /// <para>A links.</para>
23     /// </param>
24     [MethodImpl(MethodImplOptions.AggressiveInlining)]
25     public LinksCascadeUniquenessAndUsagesResolver(ILinks<TLink> links) : base(links) { }
26
27     /// <summary>
28     /// <para>
29     /// Resolves the address change conflict using the specified old link address.
30     /// </para>
31     /// </summary>
32     /// <param name="oldLinkAddress">
33     /// <para>The old link address.</para>
34     /// </param>
35     /// <param name="newLinkAddress">
36     /// <para>The new link address.</para>
37     /// </param>
38     /// <returns>
39     /// <para>The link</para>
40     /// </returns>
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     protected override TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
43     ↪ newLinkAddress)
44     {
45         // Use Facade (the last decorator) to ensure recursion working correctly
46         _facade.MergeUsages(oldLinkAddress, newLinkAddress);
47         return base.ResolveAddressChangeConflict(oldLinkAddress, newLinkAddress);
48     }
49 }
50 }
51 }
52 }
53 }
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        /// </summary>
19        /// <param name="links">
20        /// <para>A links.</para>
21        /// </param>
22        [MethodImpl(MethodImplOptions.AggressiveInlining)]
23        public LinksCascadeUsagesResolver(ILinks<TLink> links) : base(links) { }
24
25        /// <summary>
26        /// <para>
27        /// Deletes the restrictions.
28        /// </para>
29        /// </summary>
30        /// <param name="restrictions">
31        /// <para>The restrictions.</para>
32        /// </param>
33    }
34 }
35 }
36 }

```

```

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     /// <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    /// Eatches 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>
145    /// <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>
45            /// 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         ↪ = new DisposableWithMultipleCallsAllowed(Dispose);
77
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         ~LinksDisposableDecoratorBase() => Disposable.Destruct();
80
81         /// <summary>
82         /// <para>
83         /// Disposes this instance.
84         /// </para>
85         /// <para></para>
86         /// </summary>
87         [MethodImpl(MethodImplOptions.AggressiveInlining)]
88         public void Dispose() => Disposable.Dispose();
89
90         /// <summary>
91         /// <para>
92         /// Disposes the manual.
93         /// </para>
94         /// <para></para>
95         /// </summary>
96         /// <param name="manual">
97         /// <para>The manual.</para>
98         /// <para></para>
99         /// </param>
100        /// <param name="wasDisposed">
101        /// <para>The was disposed.</para>
102        /// <para></para>
103        /// </param>
104        [MethodImpl(MethodImplOptions.AggressiveInlining)]
105        protected virtual void Dispose(bool manual, bool wasDisposed)
106        {
107            if (!wasDisposed)
108            {
109                _links.DisposeIfPossible();
110            }
111        }
112    }

```

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(IList<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             return updatedLink;
96         }
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(IList<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         private struct Transition
37         {
38             /// <summary>
39             /// <para>
40             /// The before.
41             /// </para>
42             /// <para></para>
43             /// </summary>
44             public IList<TLink> Before;
45             /// <summary>
46             /// <para>
47             /// The after.
48             /// </para>
49             /// <para></para>
50             /// </summary>
51             public IList<TLink> After;
52
53             /// <summary>
54             /// <para>
55             /// Initializes a new <see cref="Transition"/> instance.
56             /// </para>
57             /// <para></para>
58             /// </summary>
59             /// <param name="before">
60             /// <para>A before.</para>

```

```

58     /// <para></para>
59     /// </param>
60     /// <param name="after">
61     /// <para>A after.</para>
62     /// <para></para>
63     /// </param>
64     public Transition(IList<TLink> before, IList<TLink> after)
65     {
66         Before = before;
67         After = after;
68     }
69 }
70
71 //public static readonly TLink NullConstant = Use<LinksConstants<TLink>>.Single.Null;
72 //public static readonly IReadOnlyList<TLink> NullLink = new
73     ↳ ReadOnlyCollection<TLink>(new List<TLink> { NullConstant, NullConstant, NullConstant
74     ↳ });
75
76 // TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
77     ↳ (Links-Expression)
78     /// <summary>
79     /// <para>
80     /// Triggers the restriction.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="restriction">
85     /// <para>The restriction.</para>
86     /// <para></para>
87     /// </param>
88     /// <param name="matchedHandler">
89     /// <para>The matched handler.</para>
90     /// <para></para>
91     /// </param>
92     /// <param name="substitution">
93     /// <para>The substitution.</para>
94     /// <para></para>
95     /// </param>
96     /// <param name="substitutedHandler">
97     /// <para>The substituted handler.</para>
98     /// <para></para>
99     /// </param>
100    /// <returns>
101    /// <para>The link</para>
102    /// <para></para>
103    /// </returns>
104    public TLink Trigger(IList<TLink> restriction, Func<IList<TLink>, IList<TLink>, TLink>
105    ↳ matchedHandler, IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
106    ↳ substitutedHandler)
107    {
108        ///List<Transition> transitions = null;
109        ///if (!restriction.IsNullOrEmpty())
110        ///{
111        ///    // Есть причина делать проход (чтение)
112        ///    if (matchedHandler != null)
113        ///    {
114        ///        if (!substitution.IsNullOrEmpty())
115        ///        {
116        ///            // restriction => { 0, 0, 0 } | { 0 } // Create
117        ///            // substitution => { itself, 0, 0 } | { itself, itself, itself } //
118        ///            ↳ Create / Update
119        ///            // substitution => { 0, 0, 0 } | { 0 } // Delete
120        ///            transitions = new List<Transition>();
121        ///            if (Equals(substitution[Constants.IndexPart], Constants.Null))
122        ///            {
123        ///                // If index is Null, that means we always ignore every other
124        ///                ↳ value (they are also Null by definition)
125        ///                var matchDecision = matchedHandler(, NullLink);
126        ///                if (Equals(matchDecision, Constants.Break))
127        ///                {
128        ///                    return false;
129        ///                }
130        ///                if (!Equals(matchDecision, Constants.Skip))
131        ///                {
132        ///                    transitions.Add(new Transition(matchedLink, newValue));
133        ///                }
134        ///            }
135        ///            else
136        ///            {
137        ///                Func<T, bool> handler;
138        ///                handler = link =>
139        ///                {

```

```

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

```

```

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

```

```

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

```

```

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>

```

```

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

```

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 =
            ↳ EqualityComparer<T>.Default;
21
22         /// <summary>
23         /// <para>.</para>
24         /// <para>.</para>
25         /// </summary>
26         /// <typeparam name="T">
27         /// <para>.</para>
28         /// <para>.</para>
29         /// </typeparam>
30         public readonly T Source;
31
32         /// <summary>
33         /// <para>.</para>
34         /// <para>.</para>
35         /// </summary>
36         /// <typeparam name="T">
37         /// <para>.</para>
38         /// <para>.</para>
39         /// </typeparam>
40         public readonly T Target;
41
42         /// <summary>
43         /// <para>.</para>
44         /// <para>.</para>
45         /// </summary>
46         /// <typeparam name="T">
47         /// <para>.</para>
48         /// <para>.</para>
49         /// </typeparam>
50         /// <param name="source">
51         /// <para>.</para>
52         /// <para>.</para>
53         /// </param>
54         /// <param name="target">
55         /// <para>.</para>
56         /// <para>.</para>
57         /// </param>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public Doublet(T source, T target)
60         {

```



```

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

```

```

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
            ↪ 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.SearchOrCreate(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 ? OUL : linksCount - amountOfDeletions;
119     for (var i = OUL; 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     /// <summary>
133     /// <para>
134     /// Deletes the links.
135     /// </para>
136     /// <para></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     {
153         if (links.Exists(linkToDelete))
154         {
155             links.EnforceResetValues(linkToDelete);
156             links.Delete(new LinkAddress<TLink>(linkToDelete));
157         }
158     }
159
160     /// <remarks>
161     /// TODO: Возможно есть очень простой способ это сделать.
162     /// (Например просто удалить файл, или изменить его размер таким образом,
163     /// чтобы удалился весь контент)
164     /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
165     /// </remarks>
166     [MethodImpl(MethodImplOptions.AggressiveInlining)]
167     public static void DeleteAll<TLink>(this ILinks<TLink> links)
168     {
169         var equalityComparer = EqualityComparer<TLink>.Default;
170         var comparer = Comparer<TLink>.Default;
171         for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
172             ↪ Arithmetic.Decrement(i))
173         {
174             links.Delete(i);
175             if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
176             {
177                 i = links.Count();
178             }
179         }
180
181         /// <summary>
182         /// <para>
183         /// Firsts the links.
184         /// </para>
185         /// <para></para>
186         /// </summary>
187         /// <typeparam name="TLink">
188         /// <para>The link.</para>
189         /// <para></para>
190         /// </typeparam>
191         /// <param name="links">
192         /// <para>The links.</para>

```

```

193     /// <para></para>
194     /// </param>
195     /// <exception cref="InvalidOperationException">
196     /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
197     /// <para></para>
198     /// </exception>
199     /// <exception cref="InvalidOperationException">
200     /// <para>В хранилище нет связей.</para>
201     /// <para></para>
202     /// </exception>
203     /// <returns>
204     /// <para>The first link.</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     public static TLink First<TLink>(this ILinks<TLink> links)
209     {
210         TLink firstLink = default;
211         var equalityComparer = EqualityComparer<TLink>.Default;
212         if (equalityComparer.Equals(links.Count(), default))
213         {
214             throw new InvalidOperationException("В хранилище нет связей.");
215         }
216         links.Each(links.Constants.Any, links.Constants.Any, link =>
217         {
218             firstLink = link[links.Constants.IndexPart];
219             return links.Constants.Break;
220         });
221         if (equalityComparer.Equals(firstLink, default))
222         {
223             throw new InvalidOperationException("В процессе поиска по хранилищу не было
224                 ↳ найдено связей.");
225         }
226         return firstLink;
227     }
228     /// <summary>
229     /// <para>
230     /// Singles the or default using the specified links.
231     /// </para>
232     /// <para></para>
233     /// </summary>
234     /// <typeparam name="TLink">
235     /// <para>The link.</para>
236     /// <para></para>
237     /// </typeparam>
238     /// <param name="links">
239     /// <para>The links.</para>
240     /// <para></para>
241     /// </param>
242     /// <param name="query">
243     /// <para>The query.</para>
244     /// <para></para>
245     /// </param>
246     /// <returns>
247     /// <para>The result.</para>
248     /// <para></para>
249     /// </returns>
250     [MethodImpl(MethodImplOptions.AggressiveInlining)]
251     public static IList<TLink> SingleOrDefault<TLink>(this ILinks<TLink> links, IList<TLink>
252     ↳ query)
253     {
254         IList<TLink> result = null;
255         var count = 0;
256         var constants = links.Constants;
257         var @continue = constants.Continue;
258         var @break = constants.Break;
259         links.Each(linkHandler, query);
260         return result;
261     }
262     TLink linkHandler(IList<TLink> link)
263     {
264         if (count == 0)
265         {
266             result = link;
267             count++;
268             return @continue;
269         }
270         else

```

```

270     {
271         result = null;
272         return @break;
273     }
274 }
275 }
276
277 #region Paths
278
279 /// <remarks>
280 /// TODO: Как так? Как то что ниже может быть корректно?
281 /// Скорее всего практически не применимо
282 /// Предполагалось, что можно было конвертировать формируемый в проходе через
283   ↳ SequenceWalker
284 /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
285 /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
286 /// </remarks>
287 [MethodImpl(MethodImplOptions.AggressiveInlining)]
288 public static bool CheckPathExistance<TLink>(this ILinks<TLink> links, params TLink[]
289   ↳ path)
290 {
291     var current = path[0];
292     //EnsureLinkExists(current, "path");
293     if (!links.Exists(current))
294     {
295         return false;
296     }
297     var equalityComparer = EqualityComparer<TLink>.Default;
298     var constants = links.Constants;
299     for (var i = 1; i < path.Length; i++)
300     {
301         var next = path[i];
302         var values = links.GetLink(current);
303         var source = values[constants.SourcePart];
304         var target = values[constants.TargetPart];
305         if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
306   ↳ next))
307         {
308             //throw new InvalidOperationException(string.Format("Невозможно выбрать
309   ↳ путь, так как и Source и Target совпадают с элементом пути {0}.", next));
310             return false;
311         }
312         if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
313   ↳ target))
314         {
315             //throw new InvalidOperationException(string.Format("Невозможно продолжить
316   ↳ путь через элемент пути {0}", next));
317             return false;
318         }
319         current = next;
320     }
321     return true;
322 }
323
324 /// <remarks>
325 /// Может потребовать дополнительного стека для PathElement's при использовании
326   ↳ SequenceWalker.
327 /// </remarks>
328 [MethodImpl(MethodImplOptions.AggressiveInlining)]
329 public static TLink GetByKeyes<TLink>(this ILinks<TLink> links, TLink root, params int[]
330   ↳ path)
331 {
332     links.EnsureLinkExists(root, "root");
333     var currentLink = root;
334     for (var i = 0; i < path.Length; i++)
335     {
336         currentLink = links.GetLink(currentLink)[path[i]];
337     }
338     return currentLink;
339 }
340
341 /// <summary>
342 /// <para>
343 /// Gets the square matrix sequence element by index using the specified links.
344 /// </para>
345 /// <para></para>
346 /// </summary>
347 /// <typeparam name="TLink">

```

```

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

```



```

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

```

```

466 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
467   ↳ (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
468   ↳ Constants.Any - любой конец, 1.. $\infty$  конкретный конец)</param>
469 /// <param name="handler">Обработчик каждой подходящей связи.</param>
470 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
471   ↳ случае.</returns>
472 [MethodImpl(MethodImplOptions.AggressiveInlining)]
473 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
474   ↳ Func<IList<TLink>, TLink> handler) => links.Each(handler, links.Constants.Any,
475   ↳ source, target);
476
477 /// <summary>
478 /// <para>
479 /// Alls the links.
480 /// </para>
481 /// <para></para>
482 /// </summary>
483 /// <typeparam name="TLink">
484 /// <para>The link.</para>
485 /// <para></para>
486 /// </typeparam>
487 /// <param name="links">
488 /// <para>The links.</para>
489 /// <para></para>
490 /// </param>
491 /// <param name="restrictions">
492 /// <para>The restrictions.</para>
493 /// <para></para>
494 /// </param>
495 /// <returns>
496 /// <para>A list of i list t link</para>
497 /// <para></para>
498 /// </returns>
499 [MethodImpl(MethodImplOptions.AggressiveInlining)]
500 public static IList<IList<TLink>> All<TLink>(this ILinks<TLink> links, params TLink[]
501   ↳ restrictions)
502 {
503     var arraySize = CheckedConverter<TLink,
504       ↳ ulong>.Default.Convert(links.Count(restrictions));
505     if (arraySize > 0)
506     {
507         var array = new IList<TLink>[arraySize];
508         var filler = new ArrayFiller<IList<TLink>, TLink>(array,
509           ↳ links.Constants.Continue);
510         links.Each(filler.AddAndReturnConstant, restrictions);
511         return array;
512     }
513     else
514     {
515         return Array.Empty<IList<TLink>>();
516     }
517 }
518
519 /// <summary>
520 /// <para>
521 /// Alls the indices using the specified links.
522 /// </para>
523 /// <para></para>
524 /// </summary>
525 /// <typeparam name="TLink">
526 /// <para>The link.</para>
527 /// <para></para>
528 /// </typeparam>
529 /// <param name="links">
530 /// <para>The links.</para>
531 /// <para></para>
532 /// </param>
533 /// <param name="restrictions">
534 /// <para>The restrictions.</para>
535 /// <para></para>
536 /// </param>
537 /// <returns>
538 /// <para>A list of t link</para>
539 /// <para></para>
540 /// </returns>
541 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

public static IList<TLink> AllIndices<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ restrictions)
{
    var arraySize = CheckedConverter<TLink,
        ↳ ulong>.Default.Convert(links.Count(restrictions));
    if (arraySize > 0)
    {
        var array = new TLink[arraySize];
        var filler = new ArrayFiller<TLink, TLink>(array, links.Constants.Continue);
        links.Each(filler.AddFirstAndReturnConstant, restrictions);
        return array;
    }
    else
    {
        return Array.Empty<TLink>();
    }
}

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

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

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

/// <summary>
/// <para>
/// Ensures the inner reference exists using the specified links.
/// </para>
/// <para></para>
/// </summary>
/// <typeparam name="TLink">
/// <para>The link.</para>

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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

```

```

1102     /// </summary>
1103     /// <typeparam name="TLink">
1104     /// <para>The link.</para>
1105     /// <para></para>
1106     /// </typeparam>
1107     /// <param name="links">
1108     /// <para>The links.</para>
1109     /// <para></para>
1110     /// </param>
1111     /// <param name="linkIndex">
1112     /// <para>The link index.</para>
1113     /// <para></para>
1114     /// </param>
1115     /// <returns>
1116     /// <para>The bool</para>
1117     /// <para></para>
1118     /// </returns>
1119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1120     public static bool AreValuesReset<TLink>(this ILinks<TLink> links, TLink linkIndex)
1121     {
1122         var nullConstant = links.Constants.Null;
1123         var equalityComparer = EqualityComparer<TLink>.Default;
1124         var link = links.GetLink(linkIndex);
1125         for (int i = 1; i < link.Count; i++)
1126         {
1127             if (!equalityComparer.Equals(link[i], nullConstant))
1128             {
1129                 return false;
1130             }
1131         }
1132         return true;
1133     }
1134
1135     // TODO: Create a universal version of this method in Platform.Data (with using of for
1136     // ↳ loop)
1137     /// <summary>
1138     /// <para>
1139     /// Resets the values using the specified links.
1140     /// </para>
1141     /// <para></para>
1142     /// </summary>
1143     /// <typeparam name="TLink">
1144     /// <para>The link.</para>
1145     /// <para></para>
1146     /// </typeparam>
1147     /// <param name="links">
1148     /// <para>The links.</para>
1149     /// <para></para>
1150     /// </param>
1151     /// <param name="linkIndex">
1152     /// <para>The link index.</para>
1153     /// <para></para>
1154     /// </param>
1155     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1156     public static void ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1157     {
1158         var nullConstant = links.Constants.Null;
1159         var updateRequest = new Link<TLink>(linkIndex, nullConstant, nullConstant);
1160         links.Update(updateRequest);
1161     }
1162
1163     // TODO: Create a universal version of this method in Platform.Data (with using of for
1164     // ↳ loop)
1165     /// <summary>
1166     /// <para>
1167     /// Enforces the reset values using the specified links.
1168     /// </para>
1169     /// <para></para>
1170     /// </summary>
1171     /// <typeparam name="TLink">
1172     /// <para>The link.</para>
1173     /// <para></para>
1174     /// </typeparam>
1175     /// <param name="links">
1176     /// <para>The links.</para>
1177     /// <para></para>
1178     /// </param>
1179     /// <param name="linkIndex">

```

```

1178 /// <para>The link index.</para>
1179 /// <para></para>
1180 /// </param>
1181 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1182 public static void EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1183 {
1184     if (!links.AreValuesReset(linkIndex))
1185     {
1186         links.ResetValues(linkIndex);
1187     }
1188 }
1189
1190 /// <summary>
1191 /// Merging two usages graphs, all children of old link moved to be children of new link
1192   ↳ or deleted.
1193 /// </summary>
1194 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1195 public static TLink MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1196   ↳ TLink newLinkIndex)
1197 {
1198     var addressToInt64Converter = CheckedConverter<TLink, long>.Default;
1199     var equalityComparer = EqualityComparer<TLink>.Default;
1200     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1201     {
1202         var constants = links.Constants;
1203         var usagesAsSourceQuery = new Link<TLink>(constants.Any, oldLinkIndex,
1204   ↳ constants.Any);
1205         var usagesAsSourceCount =
1206   ↳ addressToInt64Converter.Convert(links.Count(usagesAsSourceQuery));
1207         var usagesAsTargetQuery = new Link<TLink>(constants.Any, constants.Any,
1208   ↳ oldLinkIndex);
1209         var usagesAsTargetCount =
1210   ↳ addressToInt64Converter.Convert(links.Count(usagesAsTargetQuery));
1211         var isStandalonePoint = Point<TLink>.IsFullPoint(links.GetLink(oldLinkIndex)) &&
1212   ↳ usagesAsSourceCount == 1 && usagesAsTargetCount == 1;
1213         if (!isStandalonePoint)
1214         {
1215             var totalUsages = usagesAsSourceCount + usagesAsTargetCount;
1216             if (totalUsages > 0)
1217             {
1218                 var usages = ArrayPool.Allocate<TLink>(totalUsages);
1219                 var usagesFiller = new ArrayFiller<TLink, TLink>(usages,
1220   ↳ links.Constants.Continue);
1221                 var i = 0L;
1222                 if (usagesAsSourceCount > 0)
1223                 {
1224                     links.Each(usagesFiller.AddFirstAndReturnConstant,
1225   ↳ usagesAsSourceQuery);
1226                     for (; i < usagesAsSourceCount; i++)
1227                     {
1228                         var usage = usages[i];
1229                         if (!equalityComparer.Equals(usage, oldLinkIndex))
1230                         {
1231                             links.Update(usage, newLinkIndex, links.GetTarget(usage));
1232                         }
1233                     }
1234                 }
1235                 if (usagesAsTargetCount > 0)
1236                 {
1237                     links.Each(usagesFiller.AddFirstAndReturnConstant,
1238   ↳ usagesAsTargetQuery);
1239                     for (; i < usages.Length; i++)
1240                     {
1241                         var usage = usages[i];
1242                         if (!equalityComparer.Equals(usage, oldLinkIndex))
1243                         {
1244                             links.Update(usage, links.GetSource(usage), newLinkIndex);
1245                         }
1246                     }
1247                 }
1248                 ArrayPool.Free(usages);
1249             }
1250         }
1251     }
1252     return newLinkIndex;
1253 }
1254
1255 /// <summary>

```

```

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

```

```

1319     /// </para>
1320     /// <para></para>
1321     /// </summary>
1322     /// <typeparam name="TLink">
1323     /// <para>The link.</para>
1324     /// <para></para>
1325     /// </typeparam>
1326     /// <param name="links">
1327     /// <para>The links.</para>
1328     /// <para></para>
1329     /// </param>
1330     /// <param name="link">
1331     /// <para>The link.</para>
1332     /// <para></para>
1333     /// </param>
1334     /// <returns>
1335     /// <para>The string</para>
1336     /// <para></para>
1337     /// </returns>
1338     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1339     public static string Format<TLink>(this ILinks<TLink> links, TLink link) =>
        ↪ links.Format(links.GetLink(link));
1340 }
1341 }

```

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         private static readonly LinksConstants<TLink> _constants =
            ↪ Default<LinksConstants<TLink>>.Instance;
27         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
28         private const int Length = 3;
29
30         /// <summary>
31         /// <para>
32         /// The index.

```

```

33     /// </para>
34     /// <para></para>
35     /// </summary>
36     public readonly TLink Index;
37     /// <summary>
38     /// <para>
39     /// The source.
40     /// </para>
41     /// <para></para>
42     /// </summary>
43     public readonly TLink Source;
44     /// <summary>
45     /// <para>
46     /// The target.
47     /// </para>
48     /// <para></para>
49     /// </summary>
50     public readonly TLink Target;
51
52     /// <summary>
53     /// <para>
54     /// Initializes a new <see cref="Link"/> instance.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     /// <param name="values">
59     /// <para>A values.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     public Link(params TLink[] values) => SetValues(values, out Index, out Source, out
        ↪ Target);
64
65     /// <summary>
66     /// <para>
67     /// Initializes a new <see cref="Link"/> instance.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <param name="values">
72     /// <para>A values.</para>
73     /// <para></para>
74     /// </param>
75     [MethodImpl(MethodImplOptions.AggressiveInlining)]
76     public Link(ICollection<TLink> values) => SetValues(values, out Index, out Source, out Target);
77
78     /// <summary>
79     /// <para>
80     /// Initializes a new <see cref="Link"/> instance.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="other">
85     /// <para>A other.</para>
86     /// <para></para>
87     /// </param>
88     /// <exception cref="NotSupportedException">
89     /// <para></para>
90     /// <para></para>
91     /// </exception>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     public Link(object other)
94     {
95         if (other is Link<TLink> otherLink)
96         {
97             SetValues(ref otherLink, out Index, out Source, out Target);
98         }
99         else if (other is ICollection<TLink> otherList)
100         {
101             SetValues(otherList, out Index, out Source, out Target);
102         }
103         else
104         {
105             throw new NotSupportedException();
106         }
107     }
108
109     /// <summary>

```

```

110    /// <para>
111    /// Initializes a new <see cref="Link"/> instance.
112    /// </para>
113    /// <para></para>
114    /// </summary>
115    /// <param name="other">
116    /// <para>A other.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    public Link(ref Link<TLink> other) => SetValues(ref other, out Index, out Source, out
    ↪ Target);
121
122    /// <summary>
123    /// <para>
124    /// Initializes a new <see cref="Link"/> instance.
125    /// </para>
126    /// <para></para>
127    /// </summary>
128    /// <param name="index">
129    /// <para>A index.</para>
130    /// <para></para>
131    /// </param>
132    /// <param name="source">
133    /// <para>A source.</para>
134    /// <para></para>
135    /// </param>
136    /// <param name="target">
137    /// <para>A target.</para>
138    /// <para></para>
139    /// </param>
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    public Link(TLink index, TLink source, TLink target)
142    {
143        Index = index;
144        Source = source;
145        Target = target;
146    }
147    [MethodImpl(MethodImplOptions.AggressiveInlining)]
148    private static void SetValues(ref Link<TLink> other, out TLink index, out TLink source,
    ↪ out TLink target)
149    {
150        index = other.Index;
151        source = other.Source;
152        target = other.Target;
153    }
154    [MethodImpl(MethodImplOptions.AggressiveInlining)]
155    private static void SetValues(IList<TLink> values, out TLink index, out TLink source,
    ↪ out TLink target)
156    {
157        switch (values.Count)
158        {
159            case 3:
160                index = values[0];
161                source = values[1];
162                target = values[2];
163                break;
164            case 2:
165                index = values[0];
166                source = values[1];
167                target = default;
168                break;
169            case 1:
170                index = values[0];
171                source = default;
172                target = default;
173                break;
174            default:
175                index = default;
176                source = default;
177                target = default;
178                break;
179        }
180    }
181
182    /// <summary>
183    /// <para>
184    /// Gets the hash code.
185    /// </para>

```



```

186    /// <para></para>
187    /// </summary>
188    /// <returns>
189    /// <para>The int</para>
190    /// <para></para>
191    /// </returns>
192    [MethodImpl(MethodImplOptions.AggressiveInlining)]
193    public override int GetHashCode() => (Index, Source, Target).GetHashCode();
194
195    /// <summary>
196    /// <para>
197    /// Determines whether this instance is null.
198    /// </para>
199    /// <para></para>
200    /// </summary>
201    /// <returns>
202    /// <para>The bool</para>
203    /// <para></para>
204    /// </returns>
205    [MethodImpl(MethodImplOptions.AggressiveInlining)]
206    public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
207        && _equalityComparer.Equals(Source, _constants.Null)
208        && _equalityComparer.Equals(Target, _constants.Null);
209
210    /// <summary>
211    /// <para>
212    /// Determines whether this instance equals.
213    /// </para>
214    /// <para></para>
215    /// </summary>
216    /// <param name="other">
217    /// <para>The other.</para>
218    /// <para></para>
219    /// </param>
220    /// <returns>
221    /// <para>The bool</para>
222    /// <para></para>
223    /// </returns>
224    [MethodImpl(MethodImplOptions.AggressiveInlining)]
225    public override bool Equals(object other) => other is Link<TLink> &&
226        ↪ Equals((Link<TLink>)other);
227
228    /// <summary>
229    /// <para>
230    /// Determines whether this instance equals.
231    /// </para>
232    /// <para></para>
233    /// </summary>
234    /// <param name="other">
235    /// <para>The other.</para>
236    /// <para></para>
237    /// </param>
238    /// <returns>
239    /// <para>The bool</para>
240    /// <para></para>
241    /// </returns>
242    [MethodImpl(MethodImplOptions.AggressiveInlining)]
243    public bool Equals(Link<TLink> other) => _equalityComparer.Equals(Index, other.Index)
244        && _equalityComparer.Equals(Source, other.Source)
245        && _equalityComparer.Equals(Target, other.Target);
246
247    /// <summary>
248    /// <para>
249    /// Returns the string using the specified index.
250    /// </para>
251    /// <para></para>
252    /// </summary>
253    /// <param name="index">
254    /// <para>The index.</para>
255    /// <para></para>
256    /// </param>
257    /// <param name="source">
258    /// <para>The source.</para>
259    /// <para></para>
260    /// </param>
261    /// <param name="target">
262    /// <para>The target.</para>
263    /// <para></para>

```

```

263     /// </param>
264     /// <returns>
265     /// <para>The string</para>
266     /// <para></para>
267     /// </returns>
268     [MethodImpl(MethodImplOptions.AggressiveInlining)]
269     public static string ToString(TLink index, TLink source, TLink target) => $"({index}:
    ↳ {source}->{target})";

270
271     /// <summary>
272     /// <para>
273     /// Returns the string using the specified source.
274     /// </para>
275     /// <para></para>
276     /// </summary>
277     /// <param name="source">
278     /// <para>The source.</para>
279     /// <para></para>
280     /// </param>
281     /// <param name="target">
282     /// <para>The target.</para>
283     /// <para></para>
284     /// </param>
285     /// <returns>
286     /// <para>The string</para>
287     /// <para></para>
288     /// </returns>
289     [MethodImpl(MethodImplOptions.AggressiveInlining)]
290     public static string ToString(TLink source, TLink target) => $"({source}->{target})";

291
292     [MethodImpl(MethodImplOptions.AggressiveInlining)]
293     public static implicit operator TLink[] (Link<TLink> link) => link.ToArray();

294
295     [MethodImpl(MethodImplOptions.AggressiveInlining)]
296     public static implicit operator Link<TLink>(TLink[] linkArray) => new
    ↳ Link<TLink>(linkArray);

297
298     /// <summary>
299     /// <para>
300     /// Returns the string.
301     /// </para>
302     /// <para></para>
303     /// </summary>
304     /// <returns>
305     /// <para>The string</para>
306     /// <para></para>
307     /// </returns>
308     [MethodImpl(MethodImplOptions.AggressiveInlining)]
309     public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
    ↳ ToString(Source, Target) : ToString(Index, Source, Target);

310
311     #region IList
312
313     /// <summary>
314     /// <para>
315     /// Gets the count value.
316     /// </para>
317     /// <para></para>
318     /// </summary>
319     public int Count
320     {
321         [MethodImpl(MethodImplOptions.AggressiveInlining)]
322         get => Length;
323     }

324
325     /// <summary>
326     /// <para>
327     /// Gets the is read only value.
328     /// </para>
329     /// <para></para>
330     /// </summary>
331     public bool IsReadOnly
332     {
333         [MethodImpl(MethodImplOptions.AggressiveInlining)]
334         get => true;
335     }

336
337     /// <summary>

```

```

338     /// <para>
339     /// The not supported exception.
340     /// </para>
341     /// <para></para>
342     /// </summary>
343     public TLink this[int index]
344     {
345         [MethodImpl(MethodImplOptions.AggressiveInlining)]
346         get
347         {
348             Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
349                 ↳ nameof(index));
350             if (index == _constants.IndexPart)
351             {
352                 return Index;
353             }
354             if (index == _constants.SourcePart)
355             {
356                 return Source;
357             }
358             if (index == _constants.TargetPart)
359             {
360                 return Target;
361             }
362             throw new NotSupportedException(); // Impossible path due to
363                 ↳ Ensure.ArgumentInRange
364         }
365         [MethodImpl(MethodImplOptions.AggressiveInlining)]
366         set => throw new NotSupportedException();
367     }
368     /// <summary>
369     /// <para>
370     /// Gets the enumerator.
371     /// </para>
372     /// <para></para>
373     /// </summary>
374     /// <returns>
375     /// <para>The enumerator</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
380     /// <summary>
381     /// <para>
382     /// Gets the enumerator.
383     /// </para>
384     /// <para></para>
385     /// </summary>
386     /// <returns>
387     /// <para>An enumerator of t link</para>
388     /// <para></para>
389     /// </returns>
390     [MethodImpl(MethodImplOptions.AggressiveInlining)]
391     public IEnumerator<TLink> GetEnumerator()
392     {
393         yield return Index;
394         yield return Source;
395         yield return Target;
396     }
397     /// <summary>
398     /// <para>
399     /// Adds the item.
400     /// </para>
401     /// <para></para>
402     /// </summary>
403     /// <param name="item">
404     /// <para>The item.</para>
405     /// <para></para>
406     /// </param>
407     [MethodImpl(MethodImplOptions.AggressiveInlining)]
408     public void Add(TLink item) => throw new NotSupportedException();
409     /// <summary>
410     /// <para>
411     /// Clears this instance.
412     /// </para>
413     /// </summary>

```

```

414     /// </para>
415     /// <para></para>
416     /// </summary>
417     [MethodImpl(MethodImplOptions.AggressiveInlining)]
418     public void Clear() => throw new NotSupportedException();
419
420     /// <summary>
421     /// <para>
422     /// Determines whether this instance contains.
423     /// </para>
424     /// <para></para>
425     /// </summary>
426     /// <param name="item">
427     /// <para>The item.</para>
428     /// <para></para>
429     /// </param>
430     /// <returns>
431     /// <para>The bool</para>
432     /// <para></para>
433     /// </returns>
434     [MethodImpl(MethodImplOptions.AggressiveInlining)]
435     public bool Contains(TLink item) => IndexOf(item) >= 0;
436
437     /// <summary>
438     /// <para>
439     /// Copies the to using the specified array.
440     /// </para>
441     /// <para></para>
442     /// </summary>
443     /// <param name="array">
444     /// <para>The array.</para>
445     /// <para></para>
446     /// </param>
447     /// <param name="arrayIndex">
448     /// <para>The array index.</para>
449     /// <para></para>
450     /// </param>
451     /// <exception cref="InvalidOperationException">
452     /// <para></para>
453     /// <para></para>
454     /// </exception>
455     [MethodImpl(MethodImplOptions.AggressiveInlining)]
456     public void CopyTo(TLink[] array, int arrayIndex)
457     {
458         Ensure.OnDebug.ArgumentNotNull(array, nameof(array));
459         Ensure.OnDebug.ArgumentInRange(arrayIndex, new Range<int>(0, array.Length - 1),
460             ↪ nameof(arrayIndex));
461         if (arrayIndex + Length > array.Length)
462         {
463             throw new InvalidOperationException();
464         }
465         array[arrayIndex++] = Index;
466         array[arrayIndex++] = Source;
467         array[arrayIndex] = Target;
468     }
469
470     /// <summary>
471     /// <para>
472     /// Determines whether this instance remove.
473     /// </para>
474     /// <para></para>
475     /// </summary>
476     /// <param name="item">
477     /// <para>The item.</para>
478     /// <para></para>
479     /// </param>
480     /// <returns>
481     /// <para>The bool</para>
482     /// <para></para>
483     /// </returns>
484     [MethodImpl(MethodImplOptions.AggressiveInlining)]
485     public bool Remove(TLink item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
486
487     /// <summary>
488     /// <para>
489     /// Indexes the of using the specified item.
490     /// </para>
491     /// <para></para>

```

```

491     /// </summary>
492     /// <param name="item">
493     /// <para>The item.</para>
494     /// <para></para>
495     /// </param>
496     /// <returns>
497     /// <para>The int</para>
498     /// <para></para>
499     /// </returns>
500     [MethodImpl(MethodImplOptions.AggressiveInlining)]
501     public int IndexOf(TLink item)
502     {
503         if (_equalityComparer.Equals(Index, item))
504         {
505             return _constants.IndexPart;
506         }
507         if (_equalityComparer.Equals(Source, item))
508         {
509             return _constants.SourcePart;
510         }
511         if (_equalityComparer.Equals(Target, item))
512         {
513             return _constants.TargetPart;
514         }
515         return -1;
516     }
517
518     /// <summary>
519     /// <para>
520     /// Inserts the index.
521     /// </para>
522     /// <para></para>
523     /// </summary>
524     /// <param name="index">
525     /// <para>The index.</para>
526     /// <para></para>
527     /// </param>
528     /// <param name="item">
529     /// <para>The item.</para>
530     /// <para></para>
531     /// </param>
532     [MethodImpl(MethodImplOptions.AggressiveInlining)]
533     public void Insert(int index, TLink item) => throw new NotSupportedException();
534
535     /// <summary>
536     /// <para>
537     /// Removes the at using the specified index.
538     /// </para>
539     /// <para></para>
540     /// </summary>
541     /// <param name="index">
542     /// <para>The index.</para>
543     /// <para></para>
544     /// </param>
545     [MethodImpl(MethodImplOptions.AggressiveInlining)]
546     public void RemoveAt(int index) => throw new NotSupportedException();
547
548     [MethodImpl(MethodImplOptions.AggressiveInlining)]
549     public static bool operator ==(Link<TLink> left, Link<TLink> right) =>
550         left.Equals(right);
551
552     [MethodImpl(MethodImplOptions.AggressiveInlining)]
553     public static bool operator !=(Link<TLink> left, Link<TLink> right) => !(left == right);
554     #endregion
555 }
556 }

```

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>
49         /// <para></para>
50         /// </param>
51         /// <returns>
52         /// <para>The bool</para>
53         /// <para></para>
54         /// </returns>
55         [MethodImpl(MethodImplOptions.AggressiveInlining)]
56         public static bool IsPartialPoint<TLink>(this Link<TLink> link) =>
57             ↪ Point<TLink>.IsPartialPoint(link);
58     }
59 }

```

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      /// </summary>
26      public static readonly long SizeInBytes = Structure<LinksHeader<TLink>>.Size;
27
28      /// <summary>
29      /// <para>
30      /// The allocated links.
31      /// </para>
32      /// <para></para>
33      /// </summary>
34      public TLink AllocatedLinks;
35      /// <summary>
36      /// <para>
37      /// The reserved links.
38      /// </para>
39      /// <para></para>
40      /// </summary>
41      public TLink ReservedLinks;
42      /// <summary>
43      /// <para>
44      /// The free links.
45      /// </para>
46      /// <para></para>
47      /// </summary>
48      public TLink FreeLinks;
49      /// <summary>
50      /// <para>
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,
        ↳ FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();
139
140    [MethodImpl(MethodImplOptions.AggressiveInlining)]
141    public static bool operator ==(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
        ↳ left.Equals(right);
142
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    public static bool operator !=(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
        ↳ !(left == right);
145    }
146 }

```

1.29 ./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> :
    ↳ RecursionlessSizeBalancedTreeMethods<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="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> 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 ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
    ↳ constants, byte* linksDataParts, byte* linksIndexParts, byte* header)
    {
85          LinksDataParts = linksDataParts;
86          LinksIndexParts = linksIndexParts;
87          Header = header;
88          Break = constants.Break;
89          Continue = constants.Continue;
90      }
91  }
92

```

```

93     /// <summary>
94     /// <para>
95     /// Gets the tree root.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <returns>
100    /// <para>The link</para>
101    /// <para></para>
102    /// </returns>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected abstract 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
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,
273         ↪ secondLink.Source, secondLink.Target);
274 }
275
276 /// <summary>
277 /// <para>
278 /// Determines whether this instance first is to the right of second.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 /// <param name="first">
283 /// <para>The first.</para>
284 /// <para></para>
285 /// </param>
286 /// <param name="second">
287 /// <para>The second.</para>
288 /// <para></para>
289 /// </param>
290 /// <returns>
291 /// <para>The bool</para>
292 /// <para></para>
293 /// </returns>
294 [MethodImpl(MethodImplOptions.AggressiveInlining)]
295 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
296 {
297     ref var firstLink = ref GetLinkDataPartReference(first);
298     ref var secondLink = ref GetLinkDataPartReference(second);
299     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
300         ↪ secondLink.Source, secondLink.Target);
301 }
302
303 /// <summary>
304 /// <para>
305 /// The zero.
306 /// </para>
307 /// <para></para>
308 /// </summary>
309 public TLink this[TLink index]
310 {
311     [MethodImpl(MethodImplOptions.AggressiveInlining)]
312     get
313     {
314         var root = GetTreeRoot();
315         if (GreaterOrEqualThan(index, GetSize(root)))
316         {
317             return Zero;
318         }
319         while (!EqualToZero(root))

```

```

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;
353         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
354             ↪ node.Key < root.Key
355         {
356             root = GetLeftOrDefault(root);
357         }
358         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
359             ↪ node.Key > root.Key
360         {
361             root = GetRightOrDefault(root);
362         }
363         else // node.Key == root.Key
364         {
365             return root;
366         }
367     }
368     return Zero;
369 }
370
371 // TODO: Return indices range instead of references count
372 /// <summary>
373 /// <para>
374 /// Counts the usages using the specified link.
375 /// </para>
376 /// <para></para>
377 /// </summary>
378 /// <param name="link">
379 /// <para>The link.</para>
380 /// <para></para>
381 /// </param>
382 /// <returns>
383 /// <para>The link</para>
384 /// <para></para>
385 /// </returns>
386 [MethodImpl(MethodImplOptions.AggressiveInlining)]
387 public TLink CountUsages(TLink link)
388 {
389     var root = GetTreeRoot();
390     var total = GetSize(root);
391     var totalRightIgnore = Zero;
392     while (!EqualToZero(root))
393     {
394         var @base = GetBasePartValue(root);

```



```

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 [MethodImpl(MethodImplOptions.AggressiveInlining)]
445 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
446 {
447     var @continue = Continue;
448     if (EqualToZero(link))
449     {
450         return @continue;
451     }
452     var linkBasePart = GetBasePartValue(link);
453     var @break = Break;
454     if (GreaterThan(linkBasePart, @base))
455     {
456         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
457         {
458             return @break;
459         }
460     }
461     else if (LessThan(linkBasePart, @base))
462     {
463         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
464         {
465             return @break;
466         }
467     }
468     else //if (linkBasePart == @base)
469     {

```

```

468         if (AreEqual(handler(GetLinkValues(link)), @break))
469         {
470             return @break;
471         }
472         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
473         {
474             return @break;
475         }
476         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
477         {
478             return @break;
479         }
480     }
481     return @continue;
482 }
483
484 /// <summary>
485 /// <para>
486 /// Prints the node value using the specified node.
487 /// </para>
488 /// <para></para>
489 /// </summary>
490 /// <param name="node">
491 /// <para>The node.</para>
492 /// <para></para>
493 /// </param>
494 /// <param name="sb">
495 /// <para>The sb.</para>
496 /// <para></para>
497 /// </param>
498 [MethodImpl(MethodImplOptions.AggressiveInlining)]
499 protected override void PrintNodeValue(TLink node, StringBuilder sb)
500 {
501     ref var link = ref GetLinkDataPartReference(node);
502     sb.Append(' ');
503     sb.Append(link.Source);
504     sb.Append('-');
505     sb.Append('>');
506     sb.Append(link.Target);
507 }
508 }
509 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using 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 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 ExternalLinksSizeBalancedTreeMethodsBase<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 }

```

```

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

```

```

187     /// <returns>
188     /// <para>A ref links header of t link</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]
192     protected virtual ref LinksHeader<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 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,
        ↪ secondLink.Source, secondLink.Target);
299 }
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
        ↪ broken)
334     }

```

```

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

```

```

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 [MethodImpl(MethodImplOptions.AggressiveInlining)]
445 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
446 {
447     var @continue = Continue;
448     if (EqualToZero(link))
449     {
450         return @continue;
451     }
452     var linkBasePart = GetBasePartValue(link);
453     var @break = Break;
454     if (GreaterThan(linkBasePart, @base))
455     {
456         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
457         {
458             return @break;
459         }
460     }
461     else if (LessThan(linkBasePart, @base))
462     {
463         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
464         {
465             return @break;
466         }
467     }
468     else //if (linkBasePart == @base)
469     {
470         if (AreEqual(handler(GetLinkValues(link)), @break))
471         {
472             return @break;
473         }
474         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
475         {
476             return @break;
477         }
478         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
479         {
480             return @break;
481         }
482     }
483     return @continue;
484 }
485
486 /// <summary>
487 /// <para>
488 /// Prints the node value using the specified node.

```



```

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

```

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         /// <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     }
56 }

```

```

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 }

```

1.32 ./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> :
15        ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="ExternalLinksSourcesSizeBalancedTreeMethods"/> instance.
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,
40            ↪ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
41            ↪ 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
59            ↪ GetLinkIndexPartReference(node).LeftAsSource;

```

```

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.
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().RootAsSource;
192
193    /// <summary>
194    /// <para>
195    /// Gets the base part value using the specified link.
196    /// </para>
197    /// <para></para>
198    /// </summary>
199    /// <param name="link">
200    /// <para>The link.</para>
201    /// <para></para>
202    /// </param>
203    /// <returns>
204    /// <para>The link</para>
205    /// <para></para>
206    /// </returns>
207    [MethodImpl(MethodImplOptions.AggressiveInlining)]
208    protected override TLink GetBasePartValue(TLink link) =>
209        ↪ GetLinkDataPartReference(link).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,
        ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
        ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
235
236     /// <summary>
237     /// <para>
238     /// Determines whether this instance first is to the right of second.
239     /// </para>
240     /// <para></para>
241     /// </summary>
242     /// <param name="firstSource">
243     /// <para>The first source.</para>
244     /// <para></para>
245     /// </param>
246     /// <param name="firstTarget">
247     /// <para>The first target.</para>
248     /// <para></para>
249     /// </param>
250     /// <param name="secondSource">
251     /// <para>The second source.</para>
252     /// <para></para>
253     /// </param>
254     /// <param name="secondTarget">
255     /// <para>The second target.</para>
256     /// <para></para>
257     /// </param>
258     /// <returns>
259     /// <para>The bool</para>
260     /// <para></para>
261     /// </returns>
262     [MethodImpl(MethodImplOptions.AggressiveInlining)]
263     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
        ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
264
265     /// <summary>
266     /// <para>
267     /// Clears the node using the specified node.
268     /// </para>
269     /// <para></para>
270     /// </summary>
271     /// <param name="node">
272     /// <para>The node.</para>
273     /// <para></para>
274     /// </param>
275     [MethodImpl(MethodImplOptions.AggressiveInlining)]
276     protected override void ClearNode(TLink node)
277     {
278         ref var link = ref GetLinkIndexPartReference(node);

```



```

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

```

1.33 ./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;
62
63         /// <summary>
64         /// <para>
65         /// Gets the right reference using the specified node.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         /// <param name="node">
70         /// <para>The node.</para>
71         /// <para></para>
72         /// </param>
73         /// <returns>
74         /// <para>The ref link</para>
75         /// <para></para>
76         /// </returns>
77         [MethodImpl(MethodImplOptions.AggressiveInlining)]
78         protected override ref TLink GetRightReference(TLink node) => ref
79             ↪ GetLinkIndexPartReference(node).RightAsTarget;
80     }
81 }

```

```

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
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) =>
    ↪ 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) =>
    ↳ GetLinkIndexPartReference(node).RightAsTarget = right;
142
143 /// <summary>
144 /// <para>
145 /// Gets the size using the specified node.
146 /// </para>
147 /// <para></para>
148 /// </summary>
149 /// <param name="node">
150 /// <para>The node.</para>
151 /// <para></para>
152 /// </param>
153 /// <returns>
154 /// <para>The link</para>
155 /// <para></para>
156 /// </returns>
157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 protected override 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.
180 /// </para>
181 /// <para></para>
182 /// </summary>
183 /// <returns>
184 /// <para>The link</para>
185 /// <para></para>
186 /// </returns>
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
189
190 /// <summary>
191 /// <para>
192 /// Gets the base part value using the specified link.
193 /// </para>
194 /// <para></para>
195 /// </summary>
196 /// <param name="link">
197 /// <para>The link.</para>
198 /// <para></para>
199 /// </param>
200 /// <returns>
201 /// <para>The link</para>
202 /// <para></para>
203 /// </returns>
204 [MethodImpl(MethodImplOptions.AggressiveInlining)]
205 protected override TLink GetBasePartValue(TLink link) =>
    ↳ GetLinkDataPartReference(link).Target;
206
207 /// <summary>
208 /// <para>
209 /// Determines whether this instance first is to the left of second.
210 /// </para>
211 /// <para></para>
212 /// </summary>
213 /// <param name="firstSource">

```

```

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

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

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

```

1.34 ./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         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
41             ↳ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
42             ↳ linksDataParts, linksIndexParts, header) { }
43
44         /// <summary>
45         /// <para>
46         /// Gets the left reference using the specified node.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="node">
51         /// <para>The node.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>The ref link</para>
56         /// <para></para>
57         /// </returns>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         protected override ref TLink GetLeftReference(TLink node) => ref
60             ↳ GetLinkIndexPartReference(node).LeftAsTarget;
61
62         /// <summary>
63         /// <para>
64         /// Gets the right reference using the specified node.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68         /// <param name="node">
69         /// <para>The node.</para>
70         /// <para></para>
71         /// </param>
72         /// <returns>
73         /// <para>The ref link</para>
74         /// <para></para>
75         /// </returns>
76         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

73     protected override ref 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>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]
145    protected override void SetRight(TLink node, TLink right) =>
146        ↪ GetLinkIndexPartReference(node).RightAsTarget = right;

```

```

145     /// Gets the size using the specified node.
146     /// </para>
147     /// <para></para>
148     /// </summary>
149     /// <param name="node">
150     /// <para>The node.</para>
151     /// <para></para>
152     /// </param>
153     /// <returns>
154     /// <para>The link</para>
155     /// <para></para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override 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     /// <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         ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
178
179     /// <summary>
180     /// <para>
181     /// Gets the tree root.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <returns>
186     /// <para>The link</para>
187     /// <para></para>
188     /// </returns>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
191
192     /// <summary>
193     /// <para>
194     /// Gets the base part value using the specified link.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="link">
199     /// <para>The link.</para>
200     /// <para></para>
201     /// </param>
202     /// <returns>
203     /// <para>The link</para>
204     /// <para></para>
205     /// </returns>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override TLink GetBasePartValue(TLink link) =>
208         ↪ GetLinkDataPartReference(link).Target;
209
210     /// <summary>
211     /// <para>
212     /// Determines whether this instance first is to the left of second.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     /// <param name="firstSource">
217     /// <para>The first source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="firstTarget">
221     /// <para>The first target.</para>
222     /// <para></para>
223     /// </param>

```

```

220     /// </param>
221     /// <param name="secondSource">
222     /// <para>The second source.</para>
223     /// <para></para>
224     /// </param>
225     /// <param name="secondTarget">
226     /// <para>The second target.</para>
227     /// <para></para>
228     /// </param>
229     /// <returns>
230     /// <para>The bool</para>
231     /// <para></para>
232     /// </returns>
233     [MethodImpl(MethodImplOptions.AggressiveInlining)]
234     protected override bool FirstIsToLeftOfSecond(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         /// <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 header.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         protected readonly byte* Header;
62
63         /// <summary>
64         /// <para>
65         /// Initializes a new <see
66         ↪ cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
67         /// </para>
68         /// <para></para>
69         /// </summary>
70         /// <param name="constants">
71         /// <para>A constants.</para>
72         /// <para></para>
73         /// </param>
74         /// <param name="linksDataParts">
75         /// <para>A links data parts.</para>
76         /// <para></para>
77         /// </param>
78         /// <param name="linksIndexParts">
79         /// <para>A links index parts.</para>
80         /// <para></para>
81         /// </param>
82         /// <param name="header">
83         /// <para>A header.</para>
84         /// <para></para>
85         /// </param>

```

```

83 [MethodImpl(MethodImplOptions.AggressiveInlining)]
84 protected InternalLinksRecursionlessSizeBalancedTreeMethodsBase(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) =>
    ↳ 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 /// <returns>
303 /// <para>The zero.</para>
304 /// <para></para>
305 /// </returns>
306 [MethodImpl(MethodImplOptions.AggressiveInlining)]
307 protected TLink SearchCore(TLink root, TLink key)
308 {

```

```

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 [MethodImpl(MethodImplOptions.AggressiveInlining)]
369 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
370 {
371     var @continue = Continue;
372     if (EqualToZero(link))
373     {
374         return @continue;
375     }
376     var @break = Break;
377     if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
378     {
379         return @break;
380     }
381     if (AreEqual(handler(GetLinkValues(link)), @break))
382     {
383         return @break;
384     }
385 }

```

```

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

```

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> :
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 data parts.
46         /// </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,
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
94     /// <summary>
95     /// <para>
96     /// Gets the tree root using the specified link.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="link">
101    /// <para>The link.</para>
102    /// <para></para>
103    /// </param>
104    /// <returns>
105    /// <para>The link</para>
106    /// <para></para>
107    /// </returns>
108    [MethodImpl(MethodImplOptions.AggressiveInlining)]
109    protected abstract TLink GetTreeRoot(TLink link);
110
111    /// <summary>
112    /// <para>
113    /// Gets the base part value using the specified link.
114    /// </para>
115    /// <para></para>
116    /// </summary>
117    /// <param name="link">
118    /// <para>The link.</para>
119    /// <para></para>
120    /// </param>

```

```

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>

```



```

/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second) =>
    ↪ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));

/// <summary>
/// <para>
/// Determines whether this instance first is to the right of second.
/// </para>
/// <para></para>
/// </summary>
/// <param name="first">
/// <para>The first.</para>
/// <para></para>
/// </param>
/// <param name="second">
/// <para>The second.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>The bool</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
    ↪ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));

/// <summary>
/// <para>
/// Gets the link values using the specified link index.
/// </para>
/// <para></para>
/// </summary>
/// <param name="linkIndex">
/// <para>The link index.</para>
/// <para></para>
/// </param>
/// <returns>
/// <para>A list of t link</para>
/// <para></para>
/// </returns>
[MethodImpl(MethodImplOptions.AggressiveInlining)]
protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
{
    ref var link = ref GetLinkDataPartReference(linkIndex);
    return new Link<TLink>(linkIndex, link.Source, link.Target);
}

/// <summary>
/// <para>
/// The zero.
/// </para>
/// <para></para>
/// </summary>
public TLink this[TLink link, TLink index]
{
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    get
    {
        var root = GetTreeRoot(link);
        if (GreaterOrEqualThan(index, GetSize(root)))
        {
            return Zero;
        }
        while (!EqualToZero(root))
        {
            var left = GetLeftOrDefault(root);
            var leftSize = GetSizeOrZero(left);
            if (LessThan(index, leftSize))
            {
                root = left;
                continue;
            }
            if (AreEqual(index, leftSize))
            {
                return root;
            }
        }
    }
}

```

```

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
    ↪ (концом).
279 /// </summary>
280 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
281 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
282 /// <returns>Индекс искомой связи.</returns>
283 [MethodImpl(MethodImplOptions.AggressiveInlining)]
284 public abstract TLink Search(TLink source, TLink target);
285
286 /// <summary>
287 /// <para>
288 /// Searches the core using the specified root.
289 /// </para>
290 /// <para></para>
291 /// </summary>
292 /// <param name="root">
293 /// <para>The root.</para>
294 /// <para></para>
295 /// </param>
296 /// <param name="key">
297 /// <para>The key.</para>
298 /// <para></para>
299 /// </param>
300 /// <returns>
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     [MethodImpl(MethodImplOptions.AggressiveInlining)]
369     private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
370     {
371         var @continue = Continue;
372         if (EqualToZero(link))
373         {
374             return @continue;
375         }
376         var @break = Break;
377         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
378         {
379             return @break;
380         }
381         if (AreEqual(handler(GetLinkValues(link)), @break))
382         {
383             return @break;
384         }
385         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
386         {
387             return @break;
388         }
389         return @continue;
390     }
391
392     /// <summary>
393     /// <para>
394     /// Prints the node value using the specified node.
395     /// </para>
396     /// <para></para>
397     /// </summary>
398     /// <param name="node">
399     /// <para>The node.</para>
400     /// <para></para>
401     /// </param>
402     /// <param name="sb">
403     /// <para>The sb.</para>
404     /// <para></para>
405     /// </param>
406     [MethodImpl(MethodImplOptions.AggressiveInlining)]
407     protected override void PrintNodeValue(TLink node, StringBuilder sb)
408     {
409         ref var link = ref GetLinkDataPartReference(node);
410         sb.Append(' ');
411         sb.Append(link.Source);
412         sb.Append(' - ');
413         sb.Append(' > ');
414         sb.Append(link.Target);
415     }
416 }

```

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             ↳ UncheckedConverter<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>
37         /// The continue.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         protected readonly TLink Continue;
42
43         /// <summary>
44         /// <para>
45         /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <param name="constants">
50         /// <para>A constants.</para>
51         /// <para></para>
52         /// </param>
53         /// <param name="linksDataParts">
54         /// <para>A links data parts.</para>
55         /// <para></para>
56         /// </param>
57         /// <param name="linksIndexParts">
58         /// <para>A links index parts.</para>
59         /// <para></para>
60         /// </param>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         public InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants, byte*
63             ↳ linksDataParts, byte* linksIndexParts)
64         {
65             _linksDataParts = linksDataParts;
66             _linksIndexParts = linksIndexParts;
67             Break = constants.Break;
68             Continue = constants.Continue;
69         }
70
71         /// <summary>
72         /// <para>
73         /// Gets the link data part reference using the specified link.
74         /// </para>
75         /// <para></para>
76         /// </summary>
77         /// <param name="link">
78         /// <para>The link.</para>
79         /// <para></para>
80         /// </param>
81         /// <returns>
82         /// <para>A ref raw link data part of t link</para>
83         /// <para></para>
84         /// </returns>
85         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

```

154     /// </param>
155     /// <returns>
156     /// <para>The link</para>
157     /// <para></para>
158     /// </returns>
159     [MethodImpl(MethodImplOptions.AggressiveInlining)]
160     protected override TLink GetPrevious(TLink element) =>
161         ↪ GetLinkIndexPartReference(element).LeftAsSource;
162
163     /// <summary>
164     /// <para>
165     /// Gets the next using the specified element.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="element">
170     /// <para>The element.</para>
171     /// <para></para>
172     /// </param>
173     /// <returns>
174     /// <para>The link</para>
175     /// <para></para>
176     /// </returns>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override TLink GetNext(TLink element) =>
179         ↪ GetLinkIndexPartReference(element).RightAsSource;
180
181     /// <summary>
182     /// <para>
183     /// Gets the size using the specified head.
184     /// </para>
185     /// <para></para>
186     /// </summary>
187     /// <param name="head">
188     /// <para>The head.</para>
189     /// <para></para>
190     /// </param>
191     /// <returns>
192     /// <para>The link</para>
193     /// <para></para>
194     /// </returns>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     protected override TLink GetSize(TLink head) =>
197         ↪ GetLinkIndexPartReference(head).SizeAsSource;
198
199     /// <summary>
200     /// <para>
201     /// Sets the first using the specified head.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="head">
206     /// <para>The head.</para>
207     /// <para></para>
208     /// </param>
209     /// <param name="element">
210     /// <para>The element.</para>
211     /// <para></para>
212     /// </param>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     protected override void SetFirst(TLink head, TLink element) =>
215         ↪ GetLinkIndexPartReference(head).RootAsSource = element;
216
217     /// <summary>
218     /// <para>
219     /// Sets the last using the specified head.
220     /// </para>
221     /// <para></para>
222     /// </summary>
223     /// <param name="head">
224     /// <para>The head.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="element">
228     /// <para>The element.</para>
229     /// <para></para>
230     /// </param>
231     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

```

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

```

```

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

```

```

155     /// <para></para>
156     /// </returns>
157     [MethodImpl(MethodImplOptions.AggressiveInlining)]
158     protected override TLink GetSize(TLink node) =>
159         ↪ GetLinkIndexPartReference(node).SizeAsSource;
160
161     /// <summary>
162     /// <para>
163     /// Sets the size using the specified node.
164     /// </para>
165     /// </summary>
166     /// <param name="node">
167     /// <para>The node.</para>
168     /// </para>
169     /// </param>
170     /// <param name="size">
171     /// <para>The size.</para>
172     /// </para>
173     /// </param>
174     [MethodImpl(MethodImplOptions.AggressiveInlining)]
175     protected override void SetSize(TLink node, TLink size) =>
176         ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
177
178     /// <summary>
179     /// <para>
180     /// Gets the tree root using the specified link.
181     /// </para>
182     /// </summary>
183     /// <param name="link">
184     /// <para>The link.</para>
185     /// </para>
186     /// </param>
187     /// <returns>
188     /// <para>The link</para>
189     /// </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     /// </summary>
200     /// <param name="link">
201     /// <para>The link.</para>
202     /// </para>
203     /// </param>
204     /// <returns>
205     /// <para>The link</para>
206     /// </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     /// </summary>
217     /// <param name="link">
218     /// <para>The link.</para>
219     /// </para>
220     /// </param>
221     /// <returns>
222     /// <para>The link</para>
223     /// </para>
224     /// </returns>
225     [MethodImpl(MethodImplOptions.AggressiveInlining)]
226     protected override TLink GetKeyPartValue(TLink link) =>
227         ↪ GetLinkDataPartReference(link).Target;

```

```

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

```

1.39 ./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> :
15        ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="InternalLinksSourcesSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="linksDataParts">
28        /// <para>A links data parts.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="linksIndexParts">
32        /// <para>A links index parts.</para>
33        /// <para></para>
34        /// </param>

```

```

34    /// <param name="header">
35    /// <para>A header.</para>
36    /// </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>
46    /// </summary>
47    /// <param name="node">
48    /// <para>The node.</para>
49    /// </para>
50    /// </param>
51    /// <returns>
52    /// <para>The ref link</para>
53    /// </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>
63    /// </summary>
64    /// <param name="node">
65    /// <para>The node.</para>
66    /// </para>
67    /// </param>
68    /// <returns>
69    /// <para>The ref link</para>
70    /// </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>
80    /// </summary>
81    /// <param name="node">
82    /// <para>The node.</para>
83    /// </para>
84    /// </param>
85    /// <returns>
86    /// <para>The link</para>
87    /// </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>
97    /// </summary>
98    /// <param name="node">
99    /// <para>The node.</para>
100    /// </para>
101    /// </param>
102    /// <returns>
103    /// <para>The link</para>
104    /// </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) =>
    ↪ GetLinkIndexPartReference(node).LeftAsSource = left;

125
126 /// <summary>
127 /// <para>
128 /// Sets the right using the specified node.
129 /// </para>
130 /// <para></para>
131 /// </summary>
132 /// <param name="node">
133 /// <para>The node.</para>
134 /// <para></para>
135 /// </param>
136 /// <param name="right">
137 /// <para>The right.</para>
138 /// <para></para>
139 /// </param>
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 protected override void SetRight(TLink node, TLink right) =>
    ↪ GetLinkIndexPartReference(node).RightAsSource = right;

142
143 /// <summary>
144 /// <para>
145 /// Gets the size using the specified node.
146 /// </para>
147 /// <para></para>
148 /// </summary>
149 /// <param name="node">
150 /// <para>The node.</para>
151 /// <para></para>
152 /// </param>
153 /// <returns>
154 /// <para>The link</para>
155 /// <para></para>
156 /// </returns>
157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 protected override TLink GetSize(TLink node) =>
    ↪ GetLinkIndexPartReference(node).SizeAsSource;

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

176
177 /// <summary>

```

```

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

```

```

253     /// <param name="source">
254     /// <para>The source.</para>
255     /// <para></para>
256     /// </param>
257     /// <param name="target">
258     /// <para>The target.</para>
259     /// <para></para>
260     /// </param>
261     /// <returns>
262     /// <para>The link</para>
263     /// <para></para>
264     /// </returns>
265     public override TLink Search(TLink source, TLink target) =>
        ↪ SearchCore(GetTreeRoot(source), target);
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
19         ↪ cref="InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
41         ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
42         ↪ base(constants, linksDataParts, linksIndexParts, header) { }
43
44         /// <summary>
45         /// <para>
46         /// Gets the left reference using the specified node.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="node">
51         /// <para>The node.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>The ref link</para>
56         /// <para></para>
57         /// </returns>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

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

```

```

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

```

```

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

```

```

1.41 ./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> :
    ↳ InternalLinksSizeBalancedTreeMethodsBase<TLink>
15 {
16     /// <summary>
17     /// <para>
18     /// Initializes a new <see cref="InternalLinksTargetsSizeBalancedTreeMethods"/> instance.
19     /// </para>
20     /// <para></para>
21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="linksDataParts">
27     /// <para>A links data parts.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="linksIndexParts">
31     /// <para>A links index parts.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="header">
35     /// <para>A header.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
    ↳ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
    ↳ linksDataParts, linksIndexParts, header) { }
40
41     /// <summary>
42     /// <para>
43     /// Gets the left reference using the specified node.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="node">
48     /// <para>The node.</para>
49     /// <para></para>
50     /// </param>
51     /// <returns>
52     /// <para>The ref link</para>
53     /// <para></para>
54     /// </returns>
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     protected override ref TLink GetLeftReference(TLink node) => ref
    ↳ 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>

```

```

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

```

```

227
228     /// <summary>
229     /// <para>
230     /// Clears the node using the specified node.
231     /// </para>
232     /// <para></para>
233     /// </summary>
234     /// <param name="node">
235     /// <para>The node.</para>
236     /// <para></para>
237     /// </param>
238     [MethodImpl(MethodImplOptions.AggressiveInlining)]
239     protected override void ClearNode(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 }

```

1.42 ./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 /// </summary>
113
114 /// <param name="dataMemory">
115 /// <para>A data memory.</para>
116 /// </param>
117
118 /// <param name="indexMemory">
119 /// <para>A index memory.</para>
120 /// </param>
121
122 /// <param name="memoryReservationStep">
123 /// <para>A memory reservation step.</para>
124 /// </param>
125
126 /// <param name="constants">
127 /// <para>A constants.</para>
128 /// </param>
129
130 /// <param name="indexTreeType">
131 /// <para>A index tree type.</para>
132 /// </param>
133
134 /// <param name="useLinkedList">
135 /// <para>A use linked list.</para>
136 /// </param>
137
138 [MethodImpl(MethodImplOptions.AggressiveInlining)]
139 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep, LinksConstants<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></para>
163    /// </summary>
164    /// <param name="dataMemory">
165    /// <para>The data memory.</para>
166    /// <para></para>
167    /// </param>
168    /// <param name="indexMemory">
169    /// <para>The index memory.</para>
170    /// <para></para>
171    /// </param>
172    [MethodImpl(MethodImplOptions.AggressiveInlining)]
173    protected override void SetPointers(IResizableDirectMemory dataMemory,
174    ↪ IResizableDirectMemory indexMemory)
175    {
176        _linksDataParts = (byte*)dataMemory.Pointer;
177        _linksIndexParts = (byte*)indexMemory.Pointer;
178        _header = _linksIndexParts;
179        if (_useLinkedList)
180        {
181            InternalSourcesListMethods = new
182            ↪ InternalLinksSourcesLinkedListMethods<TLink>(Constants, _linksDataParts,
183            ↪ _linksIndexParts);
184        }
185        else
186        {
187            InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
188        }
189        ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
190        InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
191        ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
192        UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_linksDataParts, _header);
193    }
194
195    /// <summary>
196    /// <para>
197    /// Resets the pointers.
198    /// </para>
199    /// <para></para>
200    /// </summary>
201    [MethodImpl(MethodImplOptions.AggressiveInlining)]
202    protected override void ResetPointers()
203    {
204        base.ResetPointers();
205        _linksDataParts = null;
206        _linksIndexParts = null;
207        _header = null;
208    }
209
210    /// <summary>
211    /// <para>
212    /// Gets the header reference.
213    /// </para>
214    /// <para></para>
215    /// </summary>
216    /// <returns>
217    /// <para>A ref links header of t link</para>
218    /// <para></para>
219    /// </returns>
220    [MethodImpl(MethodImplOptions.AggressiveInlining)]
221    protected override ref LinksHeader<TLink> GetHeaderReference() => ref
222    ↪ AsRef<LinksHeader<TLink>>(_header);
223
224    /// <summary>
225    /// <para>
226    /// Gets the link data part reference 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 ref raw link data part of t link</para>
236    /// <para></para>
237    /// </returns>
238    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

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          private static readonly TLink _zero = default;
33          private static readonly TLink _one = Arithmetic.Increment(_zero);
34
35          /// <summary>Возвращает размер одной связи в байтах.</summary>
36          /// <remarks>
37          /// Используется только во вне класса, не рекомендуется использовать внутри.
38          /// Так как во вне не обязательно будет доступен unsafe C#.
39          /// </remarks>
40          public static readonly long LinkDataPartSizeInBytes = RawLinkDataPart<TLink>.SizeInBytes;
41
42          /// <summary>
43          /// <para>
44          /// The size in bytes.
45          /// </para>
46          /// <para></para>
47          /// </summary>
48          public static readonly long LinkIndexPartSizeInBytes =
49          ↪ RawLinkIndexPart<TLink>.SizeInBytes;
50
51          /// <summary>
52          /// <para>
53          /// The size in bytes.
54          /// </para>
55          /// </summary>

```

```

51     /// <para></para>
52     /// </summary>
53     public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
54
55     /// <summary>
56     /// <para>
57     /// The default links size step.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
62
63     /// <summary>
64     /// <para>
65     /// The data memory.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     protected readonly IResizableDirectMemory _dataMemory;
70     /// <summary>
71     /// <para>
72     /// The index memory.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     protected readonly IResizableDirectMemory _indexMemory;
77     /// <summary>
78     /// <para>
79     /// The use linked list.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     protected readonly bool _useLinkedList;
84     /// <summary>
85     /// <para>
86     /// The data memory reservation step in bytes.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     protected readonly long _dataMemoryReservationStepInBytes;
91     /// <summary>
92     /// <para>
93     /// The index memory reservation step in bytes.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     protected readonly long _indexMemoryReservationStepInBytes;
98
99     /// <summary>
100    /// <para>
101    /// The internal sources list methods.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    protected InternalLinksSourcesLinkedListMethods<TLink> InternalSourcesListMethods;
106    /// <summary>
107    /// <para>
108    /// The internal sources tree methods.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    protected ILinksTreeMethods<TLink> InternalSourcesTreeMethods;
113    /// <summary>
114    /// <para>
115    /// The external sources tree methods.
116    /// </para>
117    /// <para></para>
118    /// </summary>
119    protected ILinksTreeMethods<TLink> ExternalSourcesTreeMethods;
120    /// <summary>
121    /// <para>
122    /// The internal targets tree methods.
123    /// </para>
124    /// <para></para>
125    /// </summary>
126    protected ILinksTreeMethods<TLink> InternalTargetsTreeMethods;
127    /// <summary>
128    /// <para>
129    /// The external targets tree methods.

```

```

130     /// </para>
131     /// <para></para>
132     /// </summary>
133     protected ILinksTreeMethods<TLink> ExternalTargetsTreeMethods;
134     // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
135     // → нужно использовать не список а дерево, так как так можно быстрее проверить на
136     // → наличие связи внутри
137     /// <summary>
138     /// <para>
139     /// The unused links list methods.
140     /// </para>
141     /// <para></para>
142     /// </summary>
143     protected ILinksListMethods<TLink> UnusedLinksListMethods;
144     /// <summary>
145     /// Возвращает общее число связей находящихся в хранилище.
146     /// </summary>
147     protected virtual TLink Total
148     {
149         [MethodImpl(MethodImplOptions.AggressiveInlining)]
150         get
151         {
152             ref var header = ref GetHeaderReference();
153             return Subtract(header.AllocatedLinks, header.FreeLinks);
154         }
155     }
156     /// <summary>
157     /// <para>
158     /// Gets the constants value.
159     /// </para>
160     /// <para></para>
161     /// </summary>
162     public virtual LinksConstants<TLink> Constants
163     {
164         [MethodImpl(MethodImplOptions.AggressiveInlining)]
165         get;
166     }
167     /// <summary>
168     /// <para>
169     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
170     /// </para>
171     /// <para></para>
172     /// </summary>
173     /// <param name="dataMemory">
174     /// <para>A data memory.</para>
175     /// <para></para>
176     /// </param>
177     /// <param name="indexMemory">
178     /// <para>A index memory.</para>
179     /// <para></para>
180     /// </param>
181     /// <param name="memoryReservationStep">
182     /// <para>A memory reservation step.</para>
183     /// <para></para>
184     /// </param>
185     /// <param name="constants">
186     /// <para>A constants.</para>
187     /// <para></para>
188     /// </param>
189     /// <param name="useLinkedList">
190     /// <para>A use linked list.</para>
191     /// <para></para>
192     /// </param>
193     [MethodImpl(MethodImplOptions.AggressiveInlining)]
194     protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
195     // → indexMemory, long memoryReservationStep, LinksConstants<TLink> constants, bool
196     // → useLinkedList)
197     {
198         _dataMemory = dataMemory;
199         _indexMemory = indexMemory;
200         _dataMemoryReservationStepInBytes = memoryReservationStep * LinkDataPartSizeInBytes;
201         _indexMemoryReservationStepInBytes = memoryReservationStep *
202         // → LinkIndexPartSizeInBytes;
203         _useLinkedList = useLinkedList;
204         Constants = constants;
205     }

```

```

204
205 /// <summary>
206 /// <para>
207 /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
208 /// </para>
209 /// </summary>
210
211 /// <param name="dataMemory">
212 /// <para>A data memory.</para>
213 /// </param>
214
215 /// <param name="indexMemory">
216 /// <para>A index memory.</para>
217 /// </param>
218
219 /// <param name="memoryReservationStep">
220 /// <para>A memory reservation step.</para>
221 /// </param>
222
223 [MethodImpl(MethodImplOptions.AggressiveInlining)]
224 protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
    ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance, useLinkedList: true)
    ↪ { }
225
226 /// <summary>
227 /// <para>
228 /// Inits the data memory.
229 /// </para>
230 /// </summary>
231
232 /// <param name="dataMemory">
233 /// <para>The data memory.</para>
234 /// </param>
235
236 /// <param name="indexMemory">
237 /// <para>The index memory.</para>
238 /// </param>
239
240 [MethodImpl(MethodImplOptions.AggressiveInlining)]
241 protected virtual void Init(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↪ indexMemory)
242 {
243     // Read allocated links from header
244     if (indexMemory.ReservedCapacity < LinkHeaderSizeInBytes)
245     {
246         indexMemory.ReservedCapacity = LinkHeaderSizeInBytes;
247     }
248     SetPointers(dataMemory, indexMemory);
249     ref var header = ref GetHeaderReference();
250     var allocatedLinks = ConvertToInt64(header.AllocatedLinks);
251     // Adjust reserved capacity
252     var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
253     if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
254     {
255         minimumDataReservedCapacity = dataMemory.UsedCapacity;
256     }
257     if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
258     {
259         minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
260     }
261     var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
262     if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
263     {
264         minimumIndexReservedCapacity = indexMemory.UsedCapacity;
265     }
266     if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
267     {
268         minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
269     }
270     // Check for alignment
271     if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
272     {
273         minimumDataReservedCapacity = ((minimumDataReservedCapacity /
    ↪ _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
    ↪ _dataMemoryReservationStepInBytes;
274     }
275     if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)

```

```

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

```

```

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

```



```

419 else //if(source != Any && target != Any)
420 {
421     // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
422     TLink link;
423     if (externalReferencesRange.HasValue)
424     {
425         if (externalReferencesRange.Value.Contains(source) &&
426             ↳ externalReferencesRange.Value.Contains(target))
427         {
428             link = ExternalSourcesTreeMethods.Search(source, target);
429         }
430         else if (externalReferencesRange.Value.Contains(source))
431         {
432             link = InternalTargetsTreeMethods.Search(source, target);
433         }
434         else if (externalReferencesRange.Value.Contains(target))
435         {
436             if (_useLinkedList)
437             {
438                 link = ExternalSourcesTreeMethods.Search(source, target);
439             }
440             else
441             {
442                 link = InternalSourcesTreeMethods.Search(source, target);
443             }
444         }
445         else
446         {
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     }
459     }
460     }
461     }
462     }
463     }
464     }
465     }
466     }
467     }
468     }
469     }
470     }
471     }
472     }
473     }
474     }
475     }
476     }
477     }
478     }
479     }
480     }
481     }
482     }
483     }
484     }
485     }
486     }
487     }
488     }
489     }
490     }

```

```

491         {
492             value = target;
493         }
494         if (AreEqual(target, any))
495         {
496             value = source;
497         }
498         if (AreEqual(storedLinkValue.Source, value) ||
499             ↪ AreEqual(storedLinkValue.Target, value))
500         {
501             return GetOne();
502         }
503         return GetZero();
504     }
505     throw new NotSupportedException("Другие размеры и способы ограничений не
506     ↪ поддерживаются.");
507 }
508 /// <summary>
509 /// <para>
510 /// Eaches the handler.
511 /// </para>
512 /// <para></para>
513 /// </summary>
514 /// <param name="handler">
515 /// <para>The handler.</para>
516 /// <para></para>
517 /// </param>
518 /// <param name="restrictions">
519 /// <para>The restrictions.</para>
520 /// <para></para>
521 /// </param>
522 /// <exception cref="NotSupportedException">
523 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
524 /// <para></para>
525 /// </exception>
526 /// <returns>
527 /// <para>The link</para>
528 /// <para></para>
529 /// </returns>
530 [MethodImpl(MethodImplOptions.AggressiveInlining)]
531 public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
532 {
533     var constants = Constants;
534     var @break = constants.Break;
535     if (restrictions.Count == 0)
536     {
537         for (var link = GetOne(); LessOrEqualThan(link,
538             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
539         {
540             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
541             {
542                 return @break;
543             }
544         }
545         return @break;
546     }
547     var @continue = constants.Continue;
548     var any = constants.Any;
549     var index = 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         {

```

```

566         if (AreEqual(value, any))
567         {
568             return Each(handler, Array.Empty<TLink>());
569         }
570         if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
571         {
572             return @break;
573         }
574         return Each(handler, new Link<TLink>(index, any, value));
575     }
576     else
577     {
578         if (!Exists(index))
579         {
580             return @continue;
581         }
582         if (AreEqual(value, any))
583         {
584             return handler(GetLinkStruct(index));
585         }
586         ref var storedLinkValue = ref GetLinkDataPartReference(index);
587         if (AreEqual(storedLinkValue.Source, value) ||
588             AreEqual(storedLinkValue.Target, value))
589         {
590             return handler(GetLinkStruct(index));
591         }
592         return @continue;
593     }
594 }
595 if (restrictions.Count == 3)
596 {
597     var externalReferencesRange = constants.ExternalReferencesRange;
598     var source = restrictions[constants.SourcePart];
599     var target = restrictions[constants.TargetPart];
600     if (AreEqual(index, any))
601     {
602         if (AreEqual(source, any) && AreEqual(target, any))
603         {
604             return Each(handler, Array.Empty<TLink>());
605         }
606         else if (AreEqual(source, any))
607         {
608             if (externalReferencesRange.HasValue &&
609                 ⇨ externalReferencesRange.Value.Contains(target))
610             {
611                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
612             }
613             else
614             {
615                 return InternalTargetsTreeMethods.EachUsage(target, handler);
616             }
617         }
618         else if (AreEqual(target, any))
619         {
620             if (externalReferencesRange.HasValue &&
621                 ⇨ externalReferencesRange.Value.Contains(source))
622             {
623                 return ExternalSourcesTreeMethods.EachUsage(source, handler);
624             }
625             else
626             {
627                 if (_useLinkedList)
628                 {
629                     return InternalSourcesListMethods.EachUsage(source, handler);
630                 }
631                 else
632                 {
633                     return InternalSourcesTreeMethods.EachUsage(source, handler);
634                 }
635             }
636         }
637         else //if(source != Any && target != Any)
638         {
639             TLink link;
640             if (externalReferencesRange.HasValue)
641             {
642                 if (externalReferencesRange.Value.Contains(source) &&
643                     ⇨ externalReferencesRange.Value.Contains(target))

```

```

641         {
642             link = ExternalSourcesTreeMethods.Search(source, target);
643         }
644         else if (externalReferencesRange.Value.Contains(source))
645         {
646             link = InternalTargetsTreeMethods.Search(source, target);
647         }
648         else if (externalReferencesRange.Value.Contains(target))
649         {
650             if (_useLinkedList)
651             {
652                 link = ExternalSourcesTreeMethods.Search(source, target);
653             }
654             else
655             {
656                 link = InternalSourcesTreeMethods.Search(source, target);
657             }
658         }
659         else
660         {
661             if (_useLinkedList ||
662                 ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
663                 ↪ InternalTargetsTreeMethods.CountUsages(target)))
664             {
665                 link = InternalTargetsTreeMethods.Search(source, target);
666             }
667             else
668             {
669                 link = InternalSourcesTreeMethods.Search(source, target);
670             }
671         }
672     }
673     else
674     {
675         if (_useLinkedList ||
676             ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
677             ↪ InternalTargetsTreeMethods.CountUsages(target)))
678         {
679             link = InternalTargetsTreeMethods.Search(source, target);
680         }
681         else
682         {
683             link = InternalSourcesTreeMethods.Search(source, target);
684         }
685     }
686     return AreEqual(link, constants.Null) ? @continue :
687         ↪ handler(GetLinkStruct(link));
688 }
689 }
690 else
691 {
692     if (!Exists(index))
693     {
694         return @continue;
695     }
696     if (AreEqual(source, any) && AreEqual(target, any))
697     {
698         return handler(GetLinkStruct(index));
699     }
700     ref var storedLinkValue = ref GetLinkDataPartReference(index);
701     if (!AreEqual(source, any) && !AreEqual(target, any))
702     {
703         if (AreEqual(storedLinkValue.Source, source) &&
704             AreEqual(storedLinkValue.Target, target))
705         {
706             return handler(GetLinkStruct(index));
707         }
708         return @continue;
709     }
710     var value = default(TLink);
711     if (AreEqual(source, any))
712     {
713         value = target;
714     }
715     if (AreEqual(target, any))
716     {
717         value = source;
718     }

```

```

714         if (AreEqual(storedLinkValue.Source, value) ||
715             AreEqual(storedLinkValue.Target, value))
716         {
717             return handler(GetLinkStruct(index));
718         }
719         return @continue;
720     }
721 }
722 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↳ поддерживаются.");
723 }
724
725 /// <remarks>
726 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
    ↳ в другом месте (но не в менеджере памяти, а в логике Links)
727 /// </remarks>
728 [MethodImpl(MethodImplOptions.AggressiveInlining)]
729 public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
730 {
731     var constants = Constants;
732     var @null = constants.Null;
733     var externalReferencesRange = constants.ExternalReferencesRange;
734     var linkIndex = restrictions[constants.IndexPart];
735     ref var link = ref GetLinkDataPartReference(linkIndex);
736     var source = link.Source;
737     var target = link.Target;
738     ref var header = ref GetHeaderReference();
739     ref var rootAsSource = ref header.RootAsSource;
740     ref var rootAsTarget = ref header.RootAsTarget;
741     // Будет корректно работать только в том случае, если пространство выделенной связи
    ↳ предварительно заполнено нулями
742     if (!AreEqual(source, @null))
743     {
744         if (externalReferencesRange.HasValue &&
745             ↳ externalReferencesRange.Value.Contains(source))
746         {
747             ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
748         }
749         else
750         {
751             if (_useLinkedList)
752             {
753                 InternalSourcesListMethods.Detach(source, linkIndex);
754             }
755             else
756             {
757                 InternalSourcesTreeMethods.Detach(ref
758                     ↳ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
759             }
760         }
761     }
762     if (!AreEqual(target, @null))
763     {
764         if (externalReferencesRange.HasValue &&
765             ↳ externalReferencesRange.Value.Contains(target))
766         {
767             ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
768         }
769         else
770         {
771             InternalTargetsTreeMethods.Detach(ref
772                 ↳ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
773         }
774     }
775     source = link.Source = substitution[constants.SourcePart];
776     target = link.Target = substitution[constants.TargetPart];
777     if (!AreEqual(source, @null))
778     {
779         if (externalReferencesRange.HasValue &&
780             ↳ externalReferencesRange.Value.Contains(source))
781         {
782             ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
783         }
784         else
785         {
786             if (_useLinkedList)
787             {
788                 InternalSourcesListMethods.AttachAsLast(source, linkIndex);

```

```

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

```

```

857     }
858     else if (AreEqual(link, header.AllocatedLinks))
859     {
860         header.AllocatedLinks = Decrement(header.AllocatedLinks);
861         _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
862         _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
863         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
864         //   ↳ пока не дойдём до первой существующей связи
865         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
866         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
867             ↳ IsUnusedLink(header.AllocatedLinks))
868         {
869             UnusedLinksListMethods.Detach(header.AllocatedLinks);
870             header.AllocatedLinks = Decrement(header.AllocatedLinks);
871             _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
872             _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
873         }
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>
886 /// </param>
887 /// <returns>
888 /// <para>A list of t link</para>
889 /// <para></para>
890 /// </returns>
891 [MethodImpl(MethodImplOptions.AggressiveInlining)]
892 public IList<TLink> GetLinkStruct(TLink linkIndex)
893 {
894     ref var link = ref GetLinkDataPartReference(linkIndex);
895     return new Link<TLink>(linkIndex, link.Source, link.Target);
896 }
897
898 /// <remarks>
899 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
900 ///   ↳ адрес реально поменялся
901 ///
902 /// Указатель this.links может быть в том же месте,
903 /// так как 0-я связь не используется и имеет такой же размер как Header,
904 /// поэтому header размещается в том же месте, что и 0-я связь
905 /// </remarks>
906 [MethodImpl(MethodImplOptions.AggressiveInlining)]
907 protected abstract void SetPointers(IResizableDirectMemory dataMemory,
908     ↳ IResizableDirectMemory indexMemory);
909
910 /// <summary>
911 /// <para>
912 /// Resets the pointers.
913 /// </para>
914 /// <para></para>
915 /// </summary>
916 [MethodImpl(MethodImplOptions.AggressiveInlining)]
917 protected virtual void ResetPointers()
918 {
919     InternalSourcesListMethods = null;
920     InternalSourcesTreeMethods = null;
921     ExternalSourcesTreeMethods = null;
922     InternalTargetsTreeMethods = null;
923     ExternalTargetsTreeMethods = null;
924     UnusedLinksListMethods = null;
925 }
926
927 /// <summary>
928 /// <para>
929 /// Gets the header reference.
930 /// </para>
931 /// <para></para>
932 /// </summary>
933 /// <returns>
934 /// <para>A ref links header of t link</para>
935 /// <para></para>

```

```

932     /// </returns>
933     [MethodImpl(MethodImplOptions.AggressiveInlining)]
934     protected abstract ref LinksHeader<TLink> GetHeaderReference();
935
936     /// <summary>
937     /// <para>
938     /// Gets the link data part reference using the specified link index.
939     /// </para>
940     /// <para></para>
941     /// </summary>
942     /// <param name="linkIndex">
943     /// <para>The link index.</para>
944     /// <para></para>
945     /// </param>
946     /// <returns>
947     /// <para>A ref raw link data part of t link</para>
948     /// <para></para>
949     /// </returns>
950     [MethodImpl(MethodImplOptions.AggressiveInlining)]
951     protected abstract ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex);
952
953     /// <summary>
954     /// <para>
955     /// Gets the link index part reference using the specified link index.
956     /// </para>
957     /// <para></para>
958     /// </summary>
959     /// <param name="linkIndex">
960     /// <para>The link index.</para>
961     /// <para></para>
962     /// </param>
963     /// <returns>
964     /// <para>A ref raw link index part of t link</para>
965     /// <para></para>
966     /// </returns>
967     [MethodImpl(MethodImplOptions.AggressiveInlining)]
968     protected abstract ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
969     ↪ 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

```



```

1008     {
1009         // TODO: Reduce access to memory in different location (should be enough to use
1010         ↪ just linkIndexPart)
1011         ref var linkDataPart = ref GetLinkDataPartReference(linkIndex);
1012         ref var linkIndexPart = ref GetLinkIndexPartReference(linkIndex);
1013         return AreEqual(linkIndexPart.SizeAsTarget, default) &&
1014         ↪ !AreEqual(linkDataPart.Source, default);
1015     }
1016     else
1017     {
1018         return true;
1019     }
1020 }
1021
1022 /// <summary>
1023 /// <para>
1024 /// Gets the one.
1025 /// </para>
1026 /// <para></para>
1027 /// </summary>
1028 /// <returns>
1029 /// <para>The link</para>
1030 /// <para></para>
1031 /// </returns>
1032 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1033 protected virtual TLink GetOne() => _one;
1034
1035 /// <summary>
1036 /// <para>
1037 /// Gets the zero.
1038 /// </para>
1039 /// <para></para>
1040 /// </summary>
1041 /// <returns>
1042 /// <para>The link</para>
1043 /// <para></para>
1044 /// </returns>
1045 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1046 protected virtual TLink GetZero() => default;
1047
1048 /// <summary>
1049 /// <para>
1050 /// Determines whether this instance are equal.
1051 /// </para>
1052 /// <para></para>
1053 /// </summary>
1054 /// <param name="first">
1055 /// <para>The first.</para>
1056 /// <para></para>
1057 /// </param>
1058 /// <param name="second">
1059 /// <para>The second.</para>
1060 /// <para></para>
1061 /// </param>
1062 /// <returns>
1063 /// <para>The bool</para>
1064 /// <para></para>
1065 /// </returns>
1066 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1067 protected virtual bool AreEqual(TLink first, TLink second) =>
1068     ↪ _equalityComparer.Equals(first, second);
1069
1070 /// <summary>
1071 /// <para>
1072 /// Determines whether this instance less than.
1073 /// </para>
1074 /// <para></para>
1075 /// </summary>
1076 /// <param name="first">
1077 /// <para>The first.</para>
1078 /// <para></para>
1079 /// </param>
1080 /// <param name="second">
1081 /// <para>The second.</para>
1082 /// <para></para>
1083 /// </param>
1084 /// <returns>
1085 /// <para>The bool</para>

```

```

1083     /// <para></para>
1084     /// </returns>
1085     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1086     protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
1087         ↪ second) < 0;
1088
1089     /// <summary>
1090     /// <para>
1091     /// Determines whether this instance less or equal than.
1092     /// </para>
1093     /// <para></para>
1094     /// </summary>
1095     /// <param name="first">
1096     /// <para>The first.</para>
1097     /// <para></para>
1098     /// </param>
1099     /// <param name="second">
1100     /// <para>The second.</para>
1101     /// <para></para>
1102     /// </param>
1103     /// <returns>
1104     /// <para>The bool</para>
1105     /// <para></para>
1106     /// </returns>
1107     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1108     protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
1109         ↪ _comparer.Compare(first, second) <= 0;
1110
1111     /// <summary>
1112     /// <para>
1113     /// Determines whether this instance greater than.
1114     /// </para>
1115     /// <para></para>
1116     /// </summary>
1117     /// <param name="first">
1118     /// <para>The first.</para>
1119     /// <para></para>
1120     /// </param>
1121     /// <param name="second">
1122     /// <para>The second.</para>
1123     /// <para></para>
1124     /// </param>
1125     /// <returns>
1126     /// <para>The bool</para>
1127     /// <para></para>
1128     /// </returns>
1129     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1130     protected virtual bool GreaterThan(TLink first, TLink second) =>
1131         ↪ _comparer.Compare(first, second) > 0;
1132
1133     /// <summary>
1134     /// <para>
1135     /// Determines whether this instance greater or equal than.
1136     /// </para>
1137     /// <para></para>
1138     /// </summary>
1139     /// <param name="first">
1140     /// <para>The first.</para>
1141     /// <para></para>
1142     /// </param>
1143     /// <param name="second">
1144     /// <para>The second.</para>
1145     /// <para></para>
1146     /// </param>
1147     /// <returns>
1148     /// <para>The bool</para>
1149     /// <para></para>
1150     /// </returns>
1151     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1152     protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
1153         ↪ _comparer.Compare(first, second) >= 0;
1154
1155     /// <summary>
1156     /// <para>
1157     /// Converts the to int 64 using the specified value.
1158     /// </para>
1159     /// <para></para>
1160     /// </summary>

```

```

1157     /// <param name="value">
1158     /// <para>The value.</para>
1159     /// <para></para>
1160     /// </param>
1161     /// <returns>
1162     /// <para>The long</para>
1163     /// <para></para>
1164     /// </returns>
1165     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1166     protected virtual long ConvertToInt64(TLink value) =>
1167         ↪ _addressToInt64Converter.Convert(value);
1168
1169     /// <summary>
1170     /// <para>
1171     /// Converts the to address using the specified value.
1172     /// </para>
1173     /// <para></para>
1174     /// </summary>
1175     /// <param name="value">
1176     /// <para>The value.</para>
1177     /// <para></para>
1178     /// </param>
1179     /// <returns>
1180     /// <para>The link</para>
1181     /// <para></para>
1182     /// </returns>
1183     [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     /// <para></para>
1192     /// </summary>
1193     /// <param name="first">
1194     /// <para>The first.</para>
1195     /// <para></para>
1196     /// </param>
1197     /// <param name="second">
1198     /// <para>The second.</para>
1199     /// <para></para>
1200     /// </param>
1201     /// <returns>
1202     /// <para>The link</para>
1203     /// <para></para>
1204     /// </returns>
1205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1206     protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
1207         ↪ second);
1208
1209     /// <summary>
1210     /// <para>
1211     /// Subtracts the first.
1212     /// </para>
1213     /// <para></para>
1214     /// </summary>
1215     /// <param name="first">
1216     /// <para>The first.</para>
1217     /// <para></para>
1218     /// </param>
1219     /// <param name="second">
1220     /// <para>The second.</para>
1221     /// <para></para>
1222     /// </param>
1223     /// <returns>
1224     /// <para>The link</para>
1225     /// <para></para>
1226     /// </returns>
1227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1228     protected virtual TLink Subtract(TLink first, TLink second) =>
1229         ↪ Arithmetic<TLink>.Subtract(first, second);
1230
1231     /// <summary>
1232     /// <para>
1233     /// Increments the link.
1234     /// </para>

```

```

1231     /// <para></para>
1232     /// </summary>
1233     /// <param name="link">
1234     /// <para>The link.</para>
1235     /// <para></para>
1236     /// </param>
1237     /// <returns>
1238     /// <para>The link</para>
1239     /// <para></para>
1240     /// </returns>
1241     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1242     protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
1243
1244     /// <summary>
1245     /// <para>
1246     /// Decrements the link.
1247     /// </para>
1248     /// <para></para>
1249     /// </summary>
1250     /// <param name="link">
1251     /// <para>The link.</para>
1252     /// <para></para>
1253     /// </param>
1254     /// <returns>
1255     /// <para>The link</para>
1256     /// <para></para>
1257     /// </returns>
1258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1259     protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
1260
1261     #region Disposable
1262
1263     /// <summary>
1264     /// <para>
1265     /// Gets the allow multiple dispose calls value.
1266     /// </para>
1267     /// <para></para>
1268     /// </summary>
1269     protected override bool AllowMultipleDisposeCalls
1270     {
1271         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1272         get => true;
1273     }
1274
1275     /// <summary>
1276     /// <para>
1277     /// Disposes the manual.
1278     /// </para>
1279     /// <para></para>
1280     /// </summary>
1281     /// <param name="manual">
1282     /// <para>The manual.</para>
1283     /// <para></para>
1284     /// </param>
1285     /// <param name="wasDisposed">
1286     /// <para>The was disposed.</para>
1287     /// <para></para>
1288     /// </param>
1289     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1290     protected override void Dispose(bool manual, bool wasDisposed)
1291     {
1292         if (!wasDisposed)
1293         {
1294             ResetPointers();
1295             _dataMemory.DisposeIfPossible();
1296             _indexMemory.DisposeIfPossible();
1297         }
1298     }
1299
1300     #endregion
1301 }
1302 }

```

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         private readonly byte* _links;
24         private readonly byte* _header;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         /// <param name="links">
33         /// <para>A links.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public UnusedLinksListMethods(byte* links, byte* header)
42         {
43             _links = links;
44             _header = header;
45         }
46
47         /// <summary>
48         /// <para>
49         /// Gets the header reference.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <returns>
54         /// <para>A ref links header of t link</para>
55         /// <para></para>
56         /// </returns>
57         [MethodImpl(MethodImplOptions.AggressiveInlining)]
58         protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
59             ↳ AsRef<LinksHeader<TLink>>(_header);
60
61         /// <summary>
62         /// <para>
63         /// Gets the link data part reference using the specified link.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         /// <param name="link">
68         /// <para>The link.</para>
69         /// <para></para>
70         /// </param>
71         /// <returns>
72         /// <para>A ref raw link data part of t link</para>
73         /// <para></para>
74         /// </returns>
75         [MethodImpl(MethodImplOptions.AggressiveInlining)]
76         protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
77             ↳ AsRef<RawLinkDataPart<TLink>>(_links + (RawLinkDataPart<TLink>.SizeInBytes *
78                 ↳ _addressToInt64Converter.Convert(link)));
79
80         /// <summary>
81         /// <para>
82         /// Gets the first.
83         /// </para>

```

```

79     /// <para></para>
80     /// </summary>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
87
88     /// <summary>
89     /// <para>
90     /// Gets the last.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <returns>
95     /// <para>The link</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
100
101     /// <summary>
102     /// <para>
103     /// Gets the previous using the specified element.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="element">
108     /// <para>The element.</para>
109     /// <para></para>
110     /// </param>
111     /// <returns>
112     /// <para>The link</para>
113     /// <para></para>
114     /// </returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     protected override TLink GetPrevious(TLink element) =>
117         ↪ GetLinkDataPartReference(element).Source;
118
119     /// <summary>
120     /// <para>
121     /// Gets the next using the specified element.
122     /// </para>
123     /// <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) =>
135         ↪ GetLinkDataPartReference(element).Target;
136
137     /// <summary>
138     /// <para>
139     /// Gets the size.
140     /// </para>
141     /// <para></para>
142     /// </summary>
143     /// <returns>
144     /// <para>The link</para>
145     /// <para></para>
146     /// </returns>
147     [MethodImpl(MethodImplOptions.AggressiveInlining)]
148     protected override TLink GetSize() => GetHeaderReference().FreeLinks;
149
150     /// <summary>
151     /// <para>
152     /// Sets the first using the specified element.
153     /// </para>
154     /// <para></para>
155     /// </summary>
156     /// <param name="element">

```

```

155     /// <para>The element.</para>
156     /// <para></para>
157     /// </param>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
        ↪ element;

160
161     /// <summary>
162     /// <para>
163     /// Sets the last using the specified element.
164     /// </para>
165     /// <para></para>
166     /// </summary>
167     /// <param name="element">
168     /// <para>The element.</para>
169     /// <para></para>
170     /// </param>
171     [MethodImpl(MethodImplOptions.AggressiveInlining)]
172     protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
        ↪ element;

173
174     /// <summary>
175     /// <para>
176     /// Sets the previous using the specified element.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <param name="element">
181     /// <para>The element.</para>
182     /// <para></para>
183     /// </param>
184     /// <param name="previous">
185     /// <para>The previous.</para>
186     /// <para></para>
187     /// </param>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override void SetPrevious(TLink element, TLink previous) =>
        ↪ GetLinkDataPartReference(element).Source = previous;

190
191     /// <summary>
192     /// <para>
193     /// Sets the next using the specified element.
194     /// </para>
195     /// <para></para>
196     /// </summary>
197     /// <param name="element">
198     /// <para>The element.</para>
199     /// <para></para>
200     /// </param>
201     /// <param name="next">
202     /// <para>The next.</para>
203     /// <para></para>
204     /// </param>
205     [MethodImpl(MethodImplOptions.AggressiveInlining)]
206     protected override void SetNext(TLink element, TLink next) =>
        ↪ GetLinkDataPartReference(element).Target = next;

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

```

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>
33         /// <para></para>
34         /// </summary>
35         public TLink Source;
36
37         /// <summary>
38         /// <para>
39         /// The target.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink Target;
44
45         /// <summary>
46         /// <para>
47         /// Determines whether this instance equals.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="obj">
52         /// <para>The obj.</para>
53         /// <para></para>
54         /// </param>
55         /// <returns>
56         /// <para>The bool</para>
57         /// <para></para>
58         /// </returns>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public override bool Equals(object obj) => obj is RawLinkDataPart<TLink> link ?
61             ↪ Equals(link) : false;
62
63         /// <summary>
64         /// <para>
65         /// Determines whether this instance equals.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         /// <param name="other">
70         /// <para>The other.</para>
71         /// <para></para>
72         /// </param>
73         /// <returns>
74         /// <para>The bool</para>
75         /// <para></para>
76         /// </returns>
77         [MethodImpl(MethodImplOptions.AggressiveInlining)]
78         public bool Equals(RawLinkDataPart<TLink> other)
79             => _equalityComparer.Equals(Source, other.Source)
80             && _equalityComparer.Equals(Target, other.Target);
81
82         /// <summary>
83         /// <para>
84         /// 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>
94         ↪ right) => left.Equals(right);
95
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public static bool operator !=(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
98         ↪ right) => !(left == right);
99 }
100 }
```

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
37         /// <summary>
38         /// <para>
39         /// The left as source.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public TLink LeftAsSource;
44
45         /// <summary>
46         /// <para>
47         /// The right as source.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         public TLink RightAsSource;
52
53         /// <summary>
54         /// <para>
55         /// The size as source.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         public TLink SizeAsSource;
60
61         /// <summary>
62         /// <para>
63         /// The raw link index part.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         public TLink RawLinkIndexPart;
68     }
69 }

```

```

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

```

```

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

```

1.47 ./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
        ↪ cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> 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
        ↳ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↳ linksIndexParts, LinksHeader<TLink>* header)
65        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
66    {
67        LinksDataParts = linksDataParts;
68        LinksIndexParts = linksIndexParts;
69        Header = header;
70    }
71
72    /// <summary>
73    /// <para>
74    /// Gets the zero.
75    /// </para>
76    /// <para></para>
77    /// </summary>
78    /// <returns>
79    /// <para>The link</para>
80    /// <para></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    /// <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
198     ↪ always true for ulong
199
200     /// <summary>
201     /// <para>
202     /// Determines whether this instance less or equal than zero.
203     /// </para>
204     /// <para></para>
205     /// </summary>
206     /// <param name="value">
207     /// <para>The value.</para>
208     /// <para></para>
209     /// </param>
210     /// <returns>
211     /// <para>The bool</para>
212     /// <para></para>

```

```

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) =>
    ↪ ref LinksDataParts[link];
380
381 /// <summary>
382 /// <para>
383 /// Gets the link index part reference using the specified link.
384 /// </para>
385 /// <para></para>
386 /// </summary>
387 /// <param name="link">
388 /// <para>The link.</para>
389 /// <para></para>
390 /// </param>
391 /// <returns>
392 /// <para>A ref raw link index part of t link</para>
393 /// <para></para>
394 /// </returns>
395 [MethodImpl(MethodImplOptions.AggressiveInlining)]
396 protected override ref RawLinkIndexPart<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,
448             ↪ secondLink.Source, secondLink.Target);
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 :
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;
27         /// <summary>
28         /// <para>
29         /// The links index parts.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
34         /// <summary>
35         /// <para>
36         /// The header.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         protected new readonly LinksHeader<TLink>* Header;
41
42         /// <summary>
43         /// <para>
44         /// Initializes a new <see cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
45         ↪ instance.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <param name="constants">
50         /// <para>A constants.</para>
51         /// <para></para>
52         /// </param>
53         /// <param name="linksDataParts">
54         /// <para>A links data parts.</para>
55         /// <para></para>
56         /// </param>
57         /// <param name="linksIndexParts">
58         /// <para>A links index parts.</para>
59         /// <para></para>
60         /// </param>
61         /// <param name="header">
62         /// <para>A header.</para>
63         /// <para></para>
64         /// </param>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

64     protected UInt32ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↳ linksIndexParts, LinksHeader<TLink>* header)
65         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
66     {
67         LinksDataParts = linksDataParts;
68         LinksIndexParts = linksIndexParts;
69         Header = header;
70     }
71
72     /// <summary>
73     /// <para>
74     /// Gets the zero.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <returns>
79     /// <para>The link</para>
80     /// <para></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     /// <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 == OUL; // value is
    ↪ always >= 0 for ulong

```

```

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
253     ↪ for ulong
254
255     /// <summary>
256     /// <para>
257     /// Determines whether this instance less than.
258     /// </para>
259     /// <para></para>
260     /// </summary>
261     /// <param name="first">
262     /// <para>The first.</para>
263     /// <para></para>
264     /// </param>
265     /// <param name="second">
266     /// <para>The second.</para>
267     /// <para></para>
268     /// </param>
269     /// <returns>
270     /// <para>The bool</para>
271     /// <para></para>
272     /// </returns>
273     [MethodImpl(MethodImplOptions.AggressiveInlining)]
274     protected override bool LessThan(TLink first, TLink second) => first < second;
275
276     /// <summary>
277     /// <para>
278     /// Increments the value.
279     /// </para>
280     /// <para></para>
281     /// </summary>
282     /// <param name="value">
283     /// <para>The value.</para>
284     /// <para></para>
285     /// </param>
286     /// <returns>
287     /// <para>The link</para>
288     /// <para></para>
289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     protected override TLink Increment(TLink value) => ++value;
292
293     /// <summary>

```

```

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

```

```

371    /// <para>The link.</para>
372    /// <para></para>
373    /// </param>
374    /// <returns>
375    /// <para>A ref raw link data part of t link</para>
376    /// <para></para>
377    /// </returns>
378    [MethodImpl(MethodImplOptions.AggressiveInlining)]
379    protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
380        ↪ ref LinksDataParts[link];
381
382    /// <summary>
383    /// <para>
384    /// Gets the link index part reference using the specified link.
385    /// </para>
386    /// <para></para>
387    /// </summary>
388    /// <param name="link">
389    /// <para>The link.</para>
390    /// <para></para>
391    /// </param>
392    /// <returns>
393    /// <para>A ref raw link index part of t link</para>
394    /// <para></para>
395    /// </returns>
396    [MethodImpl(MethodImplOptions.AggressiveInlining)]
397    protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
398        ↪ ref LinksIndexParts[link];
399
400    /// <summary>
401    /// <para>
402    /// Determines whether this instance first is to the left of second.
403    /// </para>
404    /// <para></para>
405    /// </summary>
406    /// <param name="first">
407    /// <para>The first.</para>
408    /// <para></para>
409    /// </param>
410    /// <param name="second">
411    /// <para>The second.</para>
412    /// <para></para>
413    /// </param>
414    /// <returns>
415    /// <para>The bool</para>
416    /// <para></para>
417    /// </returns>
418    [MethodImpl(MethodImplOptions.AggressiveInlining)]
419    protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
420    {
421        ref var firstLink = ref LinksDataParts[first];
422        ref var secondLink = ref LinksDataParts[second];
423        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
424            ↪ secondLink.Source, secondLink.Target);
425    }
426
427    /// <summary>
428    /// <para>
429    /// Determines whether this instance first is to the right of second.
430    /// </para>
431    /// <para></para>
432    /// </summary>
433    /// <param name="first">
434    /// <para>The first.</para>
435    /// <para></para>
436    /// </param>
437    /// <param name="second">
438    /// <para>The second.</para>
439    /// <para></para>
440    /// </param>
441    /// <returns>
442    /// <para>The bool</para>
443    /// <para></para>
444    /// </returns>
445    [MethodImpl(MethodImplOptions.AggressiveInlining)]
446    protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
447    {
448        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 }

```

1.49 ./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     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
        ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
        ↪ cref="UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public
        ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<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
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 /// </param>
154 /// <returns>
155 /// <para>The link</para>
156 /// <para></para>
157 /// </returns>
158 [MethodImpl(MethodImplOptions.AggressiveInlining)]
159 protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
160
161 /// <summary>
162 /// <para>
163 /// Sets the size using the specified node.
164 /// </para>
165 /// <para></para>
166 /// </summary>
167 /// <param name="node">
168 /// <para>The node.</para>
169 /// <para></para>
170 /// </param>
171 /// <param name="size">
172 /// <para>The size.</para>
173 /// <para></para>
174 /// </param>
175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
176 protected override void SetSize(TLink node, TLink size) =>
177     ↳ LinksIndexParts[node].SizeAsSource = size;
178
179 /// <summary>
180 /// <para>
181 /// Gets the tree root.
182 /// </para>
183 /// <para></para>
184 /// </summary>
185 /// <returns>
186 /// <para>The link</para>
187 /// <para></para>
188 /// </returns>
189 [MethodImpl(MethodImplOptions.AggressiveInlining)]
190 protected override TLink GetTreeRoot() => Header->RootAsSource;
191
192 /// <summary>
193 /// <para>
194 /// Gets the base part value using the specified node.
195 /// </para>
196 /// <para></para>
197 /// </summary>
198 /// <param name="node">
199 /// <para>The node.</para>
200 /// <para></para>
201 /// </param>
202 /// <returns>
203 /// <para>The link</para>
204 /// <para></para>
205 /// </returns>
206 [MethodImpl(MethodImplOptions.AggressiveInlining)]
207 protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Source;
208
209 /// <summary>
210 /// <para>
211 /// Determines whether this instance first is to the left of second.
212 /// </para>
213 /// <para></para>
214 /// </summary>
215 /// <param name="firstSource">
216 /// <para>The first source.</para>

```

```

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

```

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         /// <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 UInt32ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
43             ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44             ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45             ↳ linksIndexParts, header) { }
46
47         /// <summary>
48         /// <para>
49         /// Gets the left reference using the specified node.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         /// <param name="node">
54         /// <para>The node.</para>
55         /// <para></para>
56         /// </param>
57         /// <returns>
58         /// <para>The ref link</para>
59         /// <para></para>
60         /// </returns>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         protected override ref TLink GetLeftReference(TLink node) => ref
63             ↳ LinksIndexParts[node].LeftAsSource;
64
65         /// <summary>
66         /// <para>
67         /// Gets the right reference using the specified node.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         /// <param name="node">
72         /// <para>The node.</para>
73         /// <para></para>
74         /// </param>
75         /// <returns>
76         /// <para>The ref link</para>
77         /// <para></para>
78         /// </returns>

```

```

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

```

```

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

```

```

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

```

1.51 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalance

```

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

```

```

9  /// <para>
10 /// Represents the int 32 external links targets recursionless size balanced tree methods.
11 /// </para>
12 /// <para></para>
13 /// </summary>
14 /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15 public unsafe class UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16     ↳ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17 {
18     /// <summary>
19     /// <para>
20     /// Initializes a new <see
21     ↳ cref="UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     /// <param name="constants">
26     /// <para>A constants.</para>
27     /// <para></para>
28     /// </param>
29     /// <param name="linksDataParts">
30     /// <para>A links data parts.</para>
31     /// <para></para>
32     /// </param>
33     /// <param name="linksIndexParts">
34     /// <para>A links index parts.</para>
35     /// <para></para>
36     /// </param>
37     /// <param name="header">
38     /// <para>A header.</para>
39     /// <para></para>
40     /// </param>
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     public
43     ↳ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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)]
81     protected override ref TLink GetRightReference(TLink node) => ref
82     ↳ LinksIndexParts[node].RightAsTarget;
83
84     /// <summary>
85     /// <para>
86     /// 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>

```



```

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

```

```

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

```

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

```

```

16 {
17     /// <summary>
18     /// <para>
19     /// Initializes a new <see cref="UInt32ExternalLinksTargetsSizeBalancedTreeMethods"/>
20     ↪ instance.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     /// <param name="constants">
25     /// <para>A constants.</para>
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 UInt32ExternalLinksTargetsSizeBalancedTreeMethods(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>

```

```

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.

```

```

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

```

```

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

```

1.53 ./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 :
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     /// <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
43     ↪ cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="constants">
48     /// <para>A constants.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="linksDataParts">
52     /// <para>A links data parts.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="linksIndexParts">
56     /// <para>A links index parts.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="header">
60     /// <para>A header.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected
65     ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
66     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
67     ↪ linksIndexParts, LinksHeader<TLink>* header)
68     : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
69     {
70     LinksDataParts = linksDataParts;
71     LinksIndexParts = linksIndexParts;
72     Header = header;
73     }
74
75     /// <summary>
76     /// <para>
77     /// Gets the zero.
78     /// </para>
79     /// <para></para>
80     /// </summary>
81     /// <returns>
82     /// <para>The link</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override TLink GetZero() => 0U;
87
88     /// <summary>
89     /// <para>
90     /// Determines whether this instance equal to zero.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="value">
95     /// <para>The value.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The bool</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override bool EqualToZero(TLink value) => value == 0U;

```

```

100
101     /// <summary>
102     /// <para>
103     /// Determines whether this instance are equal.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="first">
108     /// <para>The first.</para>
109     /// <para></para>
110     /// </param>
111     /// <param name="second">
112     /// <para>The second.</para>
113     /// <para></para>
114     /// </param>
115     /// <returns>
116     /// <para>The bool</para>
117     /// <para></para>
118     /// </returns>
119     [MethodImpl(MethodImplOptions.AggressiveInlining)]
120     protected override bool AreEqual(TLink first, TLink second) => first == second;
121
122     /// <summary>
123     /// <para>
124     /// Determines whether this instance greater than zero.
125     /// </para>
126     /// <para></para>
127     /// </summary>
128     /// <param name="value">
129     /// <para>The value.</para>
130     /// <para></para>
131     /// </param>
132     /// <returns>
133     /// <para>The bool</para>
134     /// <para></para>
135     /// </returns>
136     [MethodImpl(MethodImplOptions.AggressiveInlining)]
137     protected override bool GreaterThanZero(TLink value) => value > 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
    ↳ always true for ulong
197
198 /// <summary>
199 /// <para>
200 /// Determines whether this instance less or equal than zero.
201 /// </para>
202 /// <para></para>
203 /// </summary>
204 /// <param name="value">
205 /// <para>The value.</para>
206 /// <para></para>
207 /// </param>
208 /// <returns>
209 /// <para>The bool</para>
210 /// <para></para>
211 /// </returns>
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 protected override bool LessOrEqualThanZero(TLink value) => value == 0UL; // value is
    ↳ always >= 0 for ulong
214
215 /// <summary>
216 /// <para>
217 /// Determines whether this instance less or equal than.
218 /// </para>
219 /// <para></para>
220 /// </summary>
221 /// <param name="first">
222 /// <para>The first.</para>
223 /// <para></para>
224 /// </param>
225 /// <param name="second">
226 /// <para>The second.</para>
227 /// <para></para>
228 /// </param>
229 /// <returns>
230 /// <para>The bool</para>
231 /// <para></para>
232 /// </returns>
233 [MethodImpl(MethodImplOptions.AggressiveInlining)]
234 protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;
235
236 /// <summary>
237 /// <para>
238 /// Determines whether this instance less than zero.
239 /// </para>
240 /// <para></para>
241 /// </summary>
242 /// <param name="value">
243 /// <para>The value.</para>
244 /// <para></para>
245 /// </param>
246 /// <returns>
247 /// <para>The bool</para>
248 /// <para></para>
249 /// </returns>
250 [MethodImpl(MethodImplOptions.AggressiveInlining)]
251 protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
    ↳ for ulong
252

```

```

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

```

```

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

```

```

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) =>
425         ↪ GetKeyPartValue(first) > GetKeyPartValue(second);
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
27         /// <summary>
28         /// <para>
29         /// The links index parts.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
34
35         /// <summary>
36         /// <para>
37         /// The header.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         protected new readonly LinksHeader<TLink>* Header;
42
43         /// <summary>
44         /// <para>
45         /// Initializes a new <see cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
46         ↪ instance.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="constants">
51         /// <para>A constants.</para>
52         /// <para></para>
53         /// </param>
54         /// <param name="linksDataParts">
55         /// <para>A links data parts.</para>
56         /// <para></para>
57         /// </param>

```

```

53     /// </param>
54     /// <param name="linksIndexParts">
55     /// <para>A links index parts.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="header">
59     /// <para>A header.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
    {
64         LinksDataParts = linksDataParts;
65         LinksIndexParts = linksIndexParts;
66         Header = header;
67     }
68
69     /// <summary>
70     /// <para>
71     /// Gets the zero.
72     /// </para>
73     /// <para></para>
74     /// </summary>
75     /// <returns>
76     /// <para>The link</para>
77     /// <para></para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     protected override TLink GetZero() => 0U;
81
82     /// <summary>
83     /// <para>
84     /// Determines whether this instance equal to zero.
85     /// </para>
86     /// <para></para>
87     /// </summary>
88     /// <param name="value">
89     /// <para>The value.</para>
90     /// <para></para>
91     /// </param>
92     /// <returns>
93     /// <para>The bool</para>
94     /// <para></para>
95     /// </returns>
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     protected override bool EqualToZero(TLink value) => value == 0U;
98
99     /// <summary>
100    /// <para>
101    /// Determines whether this instance are equal.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    /// <param name="first">
106    /// <para>The first.</para>
107    /// <para></para>
108    /// </param>
109    /// <param name="second">
110    /// <para>The second.</para>
111    /// <para></para>
112    /// </param>
113    /// <returns>
114    /// <para>The bool</para>
115    /// <para></para>
116    /// </returns>
117    [MethodImpl(MethodImplOptions.AggressiveInlining)]
118    protected override bool AreEqual(TLink first, TLink second) => first == second;
119
120    /// <summary>
121    /// <para>
122    /// Determines whether this instance greater than zero.
123    /// </para>
124    /// <para></para>
125    /// </summary>
126    /// <param name="value">

```

```

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

```

```

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

```

```

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

```



```

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

```

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    /// <para></para>
13    /// </summary>
14    /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLink}"/>
15    public unsafe class UInt32InternalLinksSourcesLinkedListMethods :
16    ↪ InternalLinksSourcesLinkedListMethods<TLink>
17    {
18        private readonly RawLinkDataPart<TLink>* _linksDataParts;
19        private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
20
21        /// <summary>
22        /// <para>
23        /// Initializes a new <see cref="UInt32InternalLinksSourcesLinkedListMethods"/> instance.
24        /// </para>
25        /// <para></para>
26        /// </summary>
27        /// <param name="constants">
28        /// <para>A constants.</para>
29        /// </param>
30        /// <param name="linksDataParts">
31        /// <para>A links data parts.</para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// </param>
36        [MethodImpl(MethodImplOptions.AggressiveInlining)]
37        public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
38        ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
39        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
40        {
41            _linksDataParts = linksDataParts;
42            _linksIndexParts = linksIndexParts;
43        }
44
45        /// <summary>
46        /// <para>
47        /// Gets the link data part reference using the specified link.
48        /// </para>
49        /// <para></para>
50        /// </summary>
51        /// <param name="link">
52        /// <para>The link.</para>
53        /// </param>
54        /// <returns>
55        /// <para>A ref raw link data part of t link</para>
56        /// </returns>
57        [MethodImpl(MethodImplOptions.AggressiveInlining)]
58        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
59        ↪ ref _linksDataParts[link];
60
61        /// <summary>
62        /// <para>
63        /// Gets the link index part reference using the specified link.
64        /// </para>
65        /// <para></para>
66        /// </summary>
67        /// <param name="link">
68        /// <para>The link.</para>
69        /// </param>
70        /// <returns>
71        /// <para>A ref raw link index part of t link</para>
72        /// </returns>
73        [MethodImpl(MethodImplOptions.AggressiveInlining)]
74        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
75        ↪ ref _linksIndexParts[link];
76    }
77 }
78
79

```

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>
20        /// Initializes a new <see
21        ↪ cref="UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// </param>
28        /// <param name="linksDataParts">
29        /// <para>A links data parts.</para>
30        /// </param>
31        /// <param name="linksIndexParts">
32        /// <para>A links index parts.</para>
33        /// </param>
34        /// <param name="header">
35        /// <para>A header.</para>
36        /// </param>
37        /// </summary>
38        [MethodImpl(MethodImplOptions.AggressiveInlining)]
39        public
40        ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
41        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
42        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
43        ↪ 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        /// </param>
54        /// <returns>
55        /// <para>The ref link</para>
56        /// </returns>
57        [MethodImpl(MethodImplOptions.AggressiveInlining)]
58        protected override ref TLink GetLeftReference(TLink node) => ref
59        ↪ LinksIndexParts[node].LeftAsSource;
60
61        /// <summary>
62        /// <para>
63        /// Gets the right reference using the specified node.
64        /// </para>
65        /// <para></para>
66        /// </summary>
67        /// <param name="node">
68        /// <para>The node.</para>
69        /// </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>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    protected override void SetRight(TLink node, TLink right) =>
145    ↪ LinksIndexParts[node].RightAsSource = right;

```

```

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

```

```

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 }

```

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

```

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 :
16         ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32InternalLinksSourcesSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>

```

```

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 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    /// <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>

```



```

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

```

1.58 ./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 :
        ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↪ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public
42         ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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>

```

```

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>
122    /// <param name="left">
123    /// <para>The left.</para>
124    /// <para></para>
125    /// </param>
126    [MethodImpl(MethodImplOptions.AggressiveInlining)]
127    protected override void SetLeft(TLink node, TLink left) =>
128        ↳ LinksIndexParts[node].LeftAsTarget = left;
129
130    /// <summary>

```

```

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

```

```

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 }

```

1.59 ./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)]

```

```

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>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Source;
229
230     /// <summary>
231     /// <para>
232     /// Clears the node using the specified node.
233     /// </para>
234     /// <para></para>
235     /// </summary>
236     /// <param name="node">

```



```

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}"></seealso>
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>
117 /// </param>
118 /// <param name="useLinkedList">
119 /// <para>A use linked list.</para>
120 /// </para>
121 /// </param>
122 [MethodImpl(MethodImplOptions.AggressiveInlining)]
123 public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
    ↳ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
    ↳ memoryReservationStep, constants, useLinkedList)
124 {
125     if (indexTreeType == IndexTreeType.SizeBalancedTree)
126     {
127         _createInternalSourceTreeMethods = () => new
            ↳ UInt32InternalLinksSourcesSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
128         _createExternalSourceTreeMethods = () => new
            ↳ UInt32ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
129         _createInternalTargetTreeMethods = () => new
            ↳ UInt32InternalLinksTargetsSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
130         _createExternalTargetTreeMethods = () => new
            ↳ UInt32ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
131     }
132     else
133     {
134         _createInternalSourceTreeMethods = () => new
            ↳ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
135         _createExternalSourceTreeMethods = () => new
            ↳ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
136         _createInternalTargetTreeMethods = () => new
            ↳ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
137         _createExternalTargetTreeMethods = () => new
            ↳ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
138     }
139     Init(dataMemory, indexMemory);
140 }
141
142 /// <summary>
143 /// <para>
144 /// Sets the pointers using the specified data memory.
145 /// </para>
146 /// </summary>
147
148 /// <param name="dataMemory">
149 /// <para>The data memory.</para>
150 /// </para>
151 /// </param>
152 /// <param name="indexMemory">
153 /// <para>The index memory.</para>
154 /// </para>
155 /// </param>
156 [MethodImpl(MethodImplOptions.AggressiveInlining)]
157 protected override void SetPointers(IResizableDirectMemory dataMemory,
    ↳ IResizableDirectMemory indexMemory)
158 {
159     _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
160     _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
161     _header = (LinksHeader<TLink>*)indexMemory.Pointer;
162     if (_useLinkedList)
163     {
164         InternalSourcesListMethods = new
            ↳ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
            ↳ _linksIndexParts);
165     }
166     else
167     {
168         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();

```

```

169     }
170     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
171     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
172     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
173     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
174 }
175
176 /// <summary>
177 /// <para>
178 /// Resets the pointers.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 [MethodImpl(MethodImplOptions.AggressiveInlining)]
183 protected override void ResetPointers()
184 {
185     base.ResetPointers();
186     _linksDataParts = null;
187     _linksIndexParts = null;
188     _header = null;
189 }
190
191 /// <summary>
192 /// <para>
193 /// Gets the header reference.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <returns>
198 /// <para>A ref links header of t link</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
203
204 /// <summary>
205 /// <para>
206 /// Gets the link data part reference using the specified link index.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 /// <param name="linkIndex">
211 /// <para>The link index.</para>
212 /// <para></para>
213 /// </param>
214 /// <returns>
215 /// <para>A ref raw link data part of t link</para>
216 /// <para></para>
217 /// </returns>
218 [MethodImpl(MethodImplOptions.AggressiveInlining)]
219 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
220     => ref _linksDataParts[linkIndex];
221
222 /// <summary>
223 /// <para>
224 /// Gets the link index part reference using the specified link index.
225 /// </para>
226 /// <para></para>
227 /// </summary>
228 /// <param name="linkIndex">
229 /// <para>The link index.</para>
230 /// <para></para>
231 /// </param>
232 /// <returns>
233 /// <para>A ref raw link index part of t link</para>
234 /// <para></para>
235 /// </returns>
236 [MethodImpl(MethodImplOptions.AggressiveInlining)]
237 protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
238     linkIndex) => ref _linksIndexParts[linkIndex];
239
240 /// <summary>
241 /// <para>
242 /// Determines whether this instance are equal.
243 /// </para>
244 /// <para></para>
245 /// </summary>
246 /// <param name="first">

```

```

245    /// <para>The first.</para>
246    /// <para></para>
247    /// </param>
248    /// <param name="second">
249    /// <para>The second.</para>
250    /// <para></para>
251    /// </param>
252    /// <returns>
253    /// <para>The bool</para>
254    /// <para></para>
255    /// </returns>
256    [MethodImpl(MethodImplOptions.AggressiveInlining)]
257    protected override bool AreEqual(TLink first, TLink second) => first == second;
258
259    /// <summary>
260    /// <para>
261    /// Determines whether this instance less than.
262    /// </para>
263    /// <para></para>
264    /// </summary>
265    /// <param name="first">
266    /// <para>The first.</para>
267    /// <para></para>
268    /// </param>
269    /// <param name="second">
270    /// <para>The second.</para>
271    /// <para></para>
272    /// </param>
273    /// <returns>
274    /// <para>The bool</para>
275    /// <para></para>
276    /// </returns>
277    [MethodImpl(MethodImplOptions.AggressiveInlining)]
278    protected override bool LessThan(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 }

```

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         /// <para></para>
26         /// </summary>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public UInt32UnusedLinksListMethods(RawLinkDataPart<TLink>* links, LinksHeader<TLink>*
37         → header)
38             : base((byte*)links, (byte*)header)
39         {
40             _links = links;
41             _header = header;
42         }
43
44         /// <summary>
45         /// <para>
46         /// Gets the link data part reference using the specified link.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="link">
51         /// <para>The link.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>A ref raw link data part of t link</para>
56         /// <para></para>
57         /// </returns>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
60         → ref _links[link];
61
62         /// <summary>
63         /// <para>
64         /// Gets the header reference.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68         /// <returns>
69         /// <para>A ref links header of t link</para>
70         /// <para></para>
71         /// </returns>
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
74     }
75 }
```


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     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
16     /// <seealso cref="ILinksTreeMethods{TLink}"/>
17     public unsafe abstract class UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
18     ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<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;
27         /// <summary>
28         /// <para>
29         /// The links index parts.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
34         /// <summary>
35         /// <para>
36         /// The header.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         protected new readonly LinksHeader<TLink>* Header;
41
42         /// <summary>
43         /// <para>
44         /// Initializes a new <see
45         ↪ cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <param name="constants">
50         /// <para>A constants.</para>
51         /// <para></para>
52         /// </param>
53         /// <param name="linksDataParts">
54         /// <para>A links data parts.</para>
55         /// <para></para>
56         /// </param>
57         /// <param name="linksIndexParts">
58         /// <para>A links index parts.</para>
59         /// <para></para>
60         /// </param>
61         /// <param name="header">
62         /// <para>A header.</para>
63         /// <para></para>
64         /// </param>
65         [MethodImpl(MethodImplOptions.AggressiveInlining)]
66         protected
67         ↪ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
68         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
69         ↪ linksIndexParts, LinksHeader<TLink>* header)
70         : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
71         {
72             LinksDataParts = linksDataParts;
73             LinksIndexParts = linksIndexParts;
74             Header = header;
75         }
76
77         /// <summary>
78         /// <para>

```

```

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

```

```

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

```

```

228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
236
237     /// <summary>
238     /// <para>
239     /// Determines whether this instance less than zero.
240     /// </para>
241     /// <para></para>
242     /// </summary>
243     /// <param name="value">
244     /// <para>The value.</para>
245     /// <para></para>
246     /// </param>
247     /// <returns>
248     /// <para>The bool</para>
249     /// <para></para>
250     /// </returns>
251     [MethodImpl(MethodImplOptions.AggressiveInlining)]
252     protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
253     ↪ for ulong
254
255     /// <summary>
256     /// <para>
257     /// Determines whether this instance less than.
258     /// </para>
259     /// <para></para>
260     /// </summary>
261     /// <param name="first">
262     /// <para>The first.</para>
263     /// <para></para>
264     /// </param>
265     /// <param name="second">
266     /// <para>The second.</para>
267     /// <para></para>
268     /// </param>
269     /// <returns>
270     /// <para>The bool</para>
271     /// <para></para>
272     /// </returns>
273     [MethodImpl(MethodImplOptions.AggressiveInlining)]
274     protected override bool LessThan(ulong first, ulong second) => first < second;
275
276     /// <summary>
277     /// <para>
278     /// Increments the value.
279     /// </para>
280     /// <para></para>
281     /// </summary>
282     /// <param name="value">
283     /// <para>The value.</para>
284     /// <para></para>
285     /// </param>
286     /// <returns>
287     /// <para>The ulong</para>
288     /// <para></para>
289     /// </returns>
290     [MethodImpl(MethodImplOptions.AggressiveInlining)]
291     protected override ulong Increment(ulong value) => ++value;
292
293     /// <summary>
294     /// <para>
295     /// Decrements the value.
296     /// </para>
297     /// <para></para>
298     /// </summary>
299     /// <param name="value">
300     /// <para>The value.</para>
301     /// <para></para>
302     /// </param>
303     /// <returns>
304     /// <para>The ulong</para>
305     /// <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>

```

```

382     /// <para>
383     /// Gets the link index part reference using the specified link.
384     /// </para>
385     /// <para></para>
386     /// </summary>
387     /// <param name="link">
388     /// <para>The link.</para>
389     /// <para></para>
390     /// </param>
391     /// <returns>
392     /// <para>A ref raw link index part of t link</para>
393     /// <para></para>
394     /// </returns>
395     [MethodImpl(MethodImplOptions.AggressiveInlining)]
396     protected override ref RawLinkIndexPart<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     /// <para></para>
404     /// </summary>
405     /// <param name="first">
406     /// <para>The first.</para>
407     /// <para></para>
408     /// </param>
409     /// <param name="second">
410     /// <para>The second.</para>
411     /// <para></para>
412     /// </param>
413     /// <returns>
414     /// <para>The bool</para>
415     /// <para></para>
416     /// </returns>
417     [MethodImpl(MethodImplOptions.AggressiveInlining)]
418     protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
419     {
420         ref var firstLink = ref LinksDataParts[first];
421         ref var secondLink = ref LinksDataParts[second];
422         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
423             ↪ secondLink.Source, secondLink.Target);
424     }
425
426     /// <summary>
427     /// <para>
428     /// Determines whether this instance first is to the right of second.
429     /// </para>
430     /// <para></para>
431     /// </summary>
432     /// <param name="first">
433     /// <para>The first.</para>
434     /// <para></para>
435     /// </param>
436     /// <param name="second">
437     /// <para>The second.</para>
438     /// <para></para>
439     /// </param>
440     /// <returns>
441     /// <para>The bool</para>
442     /// <para></para>
443     /// </returns>
444     [MethodImpl(MethodImplOptions.AggressiveInlining)]
445     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
446     {
447         ref var firstLink = ref LinksDataParts[first];
448         ref var secondLink = ref LinksDataParts[second];
449         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
450             ↪ secondLink.Source, secondLink.Target);
451     }
452 }

```

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;
27        /// <summary>
28        /// <para>
29        /// The links index parts.
30        /// </para>
31        /// <para></para>
32        /// </summary>
33        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
34        /// <summary>
35        /// <para>
36        /// The header.
37        /// </para>
38        /// <para></para>
39        /// </summary>
40        protected new readonly LinksHeader<TLink>* Header;
41
42        /// <summary>
43        /// <para>
44        /// Initializes a new <see cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
45        ↪ instance.
46        /// </para>
47        /// <para></para>
48        /// </summary>
49        /// <param name="constants">
50        /// <para>A constants.</para>
51        /// <para></para>
52        /// </param>
53        /// <param name="linksDataParts">
54        /// <para>A links data parts.</para>
55        /// <para></para>
56        /// </param>
57        /// <param name="linksIndexParts">
58        /// <para>A links index parts.</para>
59        /// <para></para>
60        /// </param>
61        /// <param name="header">
62        /// <para>A header.</para>
63        /// <para></para>
64        /// </param>
65        [MethodImpl(MethodImplOptions.AggressiveInlining)]
66        protected UInt64ExternalLinksSizeBalancedTreeMethodsBase(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>

```

```

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

```



```

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

```

```

234 [MethodImpl(MethodImplOptions.AggressiveInlining)]
235 protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;
236
237 /// <summary>
238 /// <para>
239 /// Determines whether this instance less than zero.
240 /// </para>
241 /// <para></para>
242 /// </summary>
243 /// <param name="value">
244 /// <para>The value.</para>
245 /// <para></para>
246 /// </param>
247 /// <returns>
248 /// <para>The bool</para>
249 /// <para></para>
250 /// </returns>
251 [MethodImpl(MethodImplOptions.AggressiveInlining)]
252 protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
    ↳ for ulong
253
254 /// <summary>
255 /// <para>
256 /// Determines whether this instance less than.
257 /// </para>
258 /// <para></para>
259 /// </summary>
260 /// <param name="first">
261 /// <para>The first.</para>
262 /// <para></para>
263 /// </param>
264 /// <param name="second">
265 /// <para>The second.</para>
266 /// <para></para>
267 /// </param>
268 /// <returns>
269 /// <para>The bool</para>
270 /// <para></para>
271 /// </returns>
272 [MethodImpl(MethodImplOptions.AggressiveInlining)]
273 protected override bool LessThan(ulong first, ulong second) => first < second;
274
275 /// <summary>
276 /// <para>
277 /// Increments the value.
278 /// </para>
279 /// <para></para>
280 /// </summary>
281 /// <param name="value">
282 /// <para>The value.</para>
283 /// <para></para>
284 /// </param>
285 /// <returns>
286 /// <para>The ulong</para>
287 /// <para></para>
288 /// </returns>
289 [MethodImpl(MethodImplOptions.AggressiveInlining)]
290 protected override ulong Increment(ulong value) => ++value;
291
292 /// <summary>
293 /// <para>
294 /// Decrements the value.
295 /// </para>
296 /// <para></para>
297 /// </summary>
298 /// <param name="value">
299 /// <para>The value.</para>
300 /// <para></para>
301 /// </param>
302 /// <returns>
303 /// <para>The ulong</para>
304 /// <para></para>
305 /// </returns>
306 [MethodImpl(MethodImplOptions.AggressiveInlining)]
307 protected override ulong Decrement(ulong value) => --value;
308
309 /// <summary>
310 /// <para>

```

```

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

```

```

388     /// <para>The link.</para>
389     /// <para></para>
390     /// </param>
391     /// <returns>
392     /// <para>A ref raw link index part of t link</para>
393     /// <para></para>
394     /// </returns>
395     [MethodImpl(MethodImplOptions.AggressiveInlining)]
396     protected override ref RawLinkIndexPart<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     /// <para></para>
404     /// </summary>
405     /// <param name="first">
406     /// <para>The first.</para>
407     /// <para></para>
408     /// </param>
409     /// <param name="second">
410     /// <para>The second.</para>
411     /// <para></para>
412     /// </param>
413     /// <returns>
414     /// <para>The bool</para>
415     /// <para></para>
416     /// </returns>
417     [MethodImpl(MethodImplOptions.AggressiveInlining)]
418     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
419     {
420         ref var firstLink = ref LinksDataParts[first];
421         ref var secondLink = ref LinksDataParts[second];
422         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
423             ↪ secondLink.Source, secondLink.Target);
424     }
425
426     /// <summary>
427     /// <para>
428     /// Determines whether this instance first is to the right of second.
429     /// </para>
430     /// <para></para>
431     /// </summary>
432     /// <param name="first">
433     /// <para>The first.</para>
434     /// <para></para>
435     /// </param>
436     /// <param name="second">
437     /// <para>The second.</para>
438     /// <para></para>
439     /// </param>
440     /// <returns>
441     /// <para>The bool</para>
442     /// <para></para>
443     /// </returns>
444     [MethodImpl(MethodImplOptions.AggressiveInlining)]
445     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
446     {
447         ref var firstLink = ref LinksDataParts[first];
448         ref var secondLink = ref LinksDataParts[second];
449         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
450             ↪ secondLink.Source, secondLink.Target);
451     }
452 }

```

1.64 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalanced

```

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 :
    ↳ UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16  {
17      /// <summary>
18      /// <para>
19      /// Initializes a new <see
    ↳ cref="UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
20      /// </para>
21      /// <para></para>
22      /// </summary>
23      /// <param name="constants">
24      /// <para>A constants.</para>
25      /// <para></para>
26      /// </param>
27      /// <param name="linksDataParts">
28      /// <para>A links data parts.</para>
29      /// <para></para>
30      /// </param>
31      /// <param name="linksIndexParts">
32      /// <para>A links index parts.</para>
33      /// <para></para>
34      /// </param>
35      /// <param name="header">
36      /// <para>A header.</para>
37      /// <para></para>
38      /// </param>
39      [MethodImpl(MethodImplOptions.AggressiveInlining)]
40      public
    ↳ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<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>

```

```

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

```

```

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

```

1.65 ./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>

```



```

18     /// <para>
19     /// Initializes a new <see cref="UInt64ExternalLinksSourcesSizeBalancedTreeMethods"/>
    ↪ 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 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>

```

```

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

```

```

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 }

```

1.66 ./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>

```

```

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     ↪ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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>

```

```

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>

```

```

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

```

```

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     => 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 }

```

1.67 ./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>

```



```

34     /// </param>
35     /// <param name="header">
36     /// <para>A header.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public UInt64ExternalLinksTargetsSizeBalancedTreeMethods(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>

```

```

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

```

```

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 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 internal links recursionless size balanced tree methods base.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
16     public unsafe abstract class UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17         ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// The links data parts.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26
27         /// <summary>
28         /// <para>
29         /// The links index parts.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
34
35         /// <summary>
36         /// <para>
37         /// The header.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         protected new readonly LinksHeader<TLink>* Header;
42
43         /// <summary>
44         /// <para>
45         /// Initializes a new <see
46         ↪ cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
47         /// </para>

```

```

44     /// <para></para>
45     /// </summary>
46     /// <param name="constants">
47     /// <para>A constants.</para>
48     /// <para></para>
49     /// </param>
50     /// <param name="linksDataParts">
51     /// <para>A links data parts.</para>
52     /// <para></para>
53     /// </param>
54     /// <param name="linksIndexParts">
55     /// <para>A links index parts.</para>
56     /// <para></para>
57     /// </param>
58     /// <param name="header">
59     /// <para>A header.</para>
60     /// <para></para>
61     /// </param>
62     [MethodImpl(MethodImplOptions.AggressiveInlining)]
63     protected
64     ↪ UInt64 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     /// <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() => OUL;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance equal to zero.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="value">
94     /// <para>The value.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The bool</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override bool EqualToZero(ulong value) => value == OUL;
103
104    /// <summary>
105    /// <para>
106    /// Determines whether this instance are equal.
107    /// </para>
108    /// <para></para>
109    /// </summary>
110    /// <param name="first">
111    /// <para>The first.</para>
112    /// <para></para>
113    /// </param>
114    /// <param name="second">
115    /// <para>The second.</para>
116    /// <para></para>
117    /// </param>
118    /// <returns>
119    /// <para>The bool</para>
120    /// <para></para>
121    /// </returns>

```

```

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

```

```

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

```

```

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

```



```

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

```

```

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 :
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="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
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 UInt64InternalLinksSizeBalancedTreeMethodsBase(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 }

```

```

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
197     ↪ always true for ulong
198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance less or equal than zero.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="value">
206     /// <para>The value.</para>
207     /// <para></para>
208     /// </param>
209     /// <returns>
210     /// <para>The bool</para>
211     /// <para></para>
212     /// </returns>
213     [MethodImpl(MethodImplOptions.AggressiveInlining)]
214     protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
215     ↪ always >= 0 for ulong
216
217     /// <summary>
218     /// <para>
219     /// Determines whether this instance less or equal than.
220     /// </para>
221     /// <para></para>
222     /// </summary>
223     /// <param name="first">
224     /// <para>The first.</para>
225     /// <para></para>

```

```

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

```

```

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>

```

```

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 }

```

1.70 ./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 :
        ↪ InternalLinksSourcesLinkedListMethods<TLink>
16     {
17         private readonly RawLinkDataPart<TLink>* _linksDataParts;
18         private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt64InternalLinksSourcesLinkedListMethods"> instance.
23         /// </para>

```

```

24     /// <para></para>
25     /// </summary>
26     /// <param name="constants">
27     /// <para>A constants.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="linksDataParts">
31     /// <para>A links data parts.</para>
32     /// <para></para>
33     /// </param>
34     /// <param name="linksIndexParts">
35     /// <para>A links index parts.</para>
36     /// <para></para>
37     /// </param>
38     [MethodImpl(MethodImplOptions.AggressiveInlining)]
39     public UInt64InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
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 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 64 internal links sources recursionless size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
16    ↪ 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>
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     public
42     ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
43     ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
44     ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
45     ↪ linksIndexParts, header) { }
46
47     /// <summary>
48     /// <para>
49     /// Gets the left reference using the specified node.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     /// <param name="node">
54     /// <para>The node.</para>
55     /// <para></para>
56     /// </param>
57     /// <returns>
58     /// <para>The ref link</para>
59     /// <para></para>
60     /// </returns>
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     protected override ref TLink GetLeftReference(TLink node) => ref
63     ↪ LinksIndexParts[node].LeftAsSource;
64
65     /// <summary>
66     /// <para>
67     /// Gets the right reference using the specified node.
68     /// </para>
69     /// <para></para>
70     /// </summary>
71     /// <param name="node">
72     /// <para>The node.</para>
73     /// <para></para>
74     /// </param>
75     /// <returns>
76     /// <para>The ref link</para>
77     /// <para></para>
78     /// </returns>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     protected override ref TLink GetRightReference(TLink node) => ref
81     ↪ LinksIndexParts[node].RightAsSource;
82
83     /// <summary>
84     /// <para>
85     /// Gets the left using the specified node.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     /// <param name="node">
90     /// <para>The node.</para>
91     /// <para></para>
92     /// </param>
93     /// <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>

```

```

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

```

```

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/UInt64InternalLinksSourcesSizeBalancedTreeMethods

```

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"/>
        ↪ 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 UInt64InternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
        ↪ linksIndexParts, header) { }
41
42     /// <summary>

```

```

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

```

```

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

```

```

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

```

```

}

```

```

}

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16         ↳ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↳ cref="UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// </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
42         ↳ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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 ulong</para>
59         /// <para></para>
60         /// </returns>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         protected override ref ulong GetLeftReference(ulong node) => ref
63         ↳ LinksIndexParts[node].LeftAsTarget;
64
65         /// <summary>
66         /// <para>
67         /// Gets the right reference using the specified node.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         /// <param name="node">
72         /// <para>The node.</para>
73         /// <para></para>
74         /// </param>
75         /// <returns>
76         /// <para>The ref ulong</para>
77         /// <para></para>
78         /// </returns>
79         [MethodImpl(MethodImplOptions.AggressiveInlining)]
80         protected override ref ulong GetRightReference(ulong node) => ref
81         ↳ LinksIndexParts[node].RightAsTarget;
82     }
83 }

```



```

71     /// <para></para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override ref ulong GetRightReference(ulong 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    /// <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    ↪ LinksIndexParts[node].LeftAsTarget = left;
126
127    /// <summary>
128    /// <para>
129    /// Sets the right using the specified node.
130    /// </para>
131    /// <para></para>
132    /// </summary>
133    /// <param name="node">
134    /// <para>The node.</para>
135    /// <para></para>
136    /// </param>
137    /// <param name="right">
138    /// <para>The right.</para>
139    /// <para></para>
140    /// </param>
141    [MethodImpl(MethodImplOptions.AggressiveInlining)]
142    protected override void SetRight(TLink node, TLink right) =>
143    ↪ LinksIndexParts[node].RightAsTarget = right;
144
145    /// <summary>
146    /// <para>

```

```

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) =>
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>

```

```

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 }

```

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

```

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 :
16         ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
21         ↪ instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">

```

```

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

```

```

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

```

1.75 ./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
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46         /// <summary>
47         /// <para>
48         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <param name="dataMemory">
53         /// <para>A data memory.</para>
54         /// <para></para>
55         /// </param>
56         /// <param name="indexMemory">
57         /// <para>A index memory.</para>
58         /// <para></para>
59         /// </param>

```

```

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

66
67     /// <summary>
68     /// <para>
69     /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
70     /// </para>
71     /// </para>
72     /// </summary>
73     /// <param name="dataMemory">
74     /// <para>A data memory.</para>
75     /// </para>
76     /// </param>
77     /// <param name="indexMemory">
78     /// <para>A index memory.</para>
79     /// </para>
80     /// </param>
81     /// <param name="memoryReservationStep">
82     /// <para>A memory reservation step.</para>
83     /// </para>
84     /// </param>
85     /// <param name="constants">
86     /// <para>A constants.</para>
87     /// </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>
97     /// </summary>
98     /// <param name="dataMemory">
99     /// <para>A data memory.</para>
100    /// </para>
101    /// </param>
102    /// <param name="indexMemory">
103    /// <para>A index memory.</para>
104    /// </para>
105    /// </param>
106    /// <param name="memoryReservationStep">
107    /// <para>A memory reservation step.</para>
108    /// </para>
109    /// </param>
110    /// <param name="constants">
111    /// <para>A constants.</para>
112    /// </para>
113    /// </param>
114    /// <param name="indexTreeType">
115    /// <para>A index tree type.</para>
116    /// </para>
117    /// </param>
118    /// <param name="useLinkedList">
119    /// <para>A use linked list.</para>
120    /// </para>
121    /// </param>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↳ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
        ↳ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↳ memoryReservationStep, constants, useLinkedList)
124    {
125        if (indexTreeType == IndexTreeType.SizeBalancedTree)
126        {

```



```

127     _createInternalSourceTreeMethods = () => new
        ↳ UInt64InternalLinksSourcesSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
128     _createExternalSourceTreeMethods = () => new
        ↳ UInt64ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
129     _createInternalTargetTreeMethods = () => new
        ↳ UInt64InternalLinksTargetsSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
130     _createExternalTargetTreeMethods = () => new
        ↳ UInt64ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
131 }
132 else
133 {
134     _createInternalSourceTreeMethods = () => new
        ↳ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
135     _createExternalSourceTreeMethods = () => new
        ↳ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
136     _createInternalTargetTreeMethods = () => new
        ↳ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
137     _createExternalTargetTreeMethods = () => new
        ↳ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
        ↳ _linksDataParts, _linksIndexParts, _header);
138 }
139 Init(dataMemory, indexMemory);
140 }
141
142 /// <summary>
143 /// <para>
144 /// Sets the pointers using the specified data memory.
145 /// </para>
146 /// <para></para>
147 /// </summary>
148 /// <param name="dataMemory">
149 /// <para>The data memory.</para>
150 /// <para></para>
151 /// </param>
152 /// <param name="indexMemory">
153 /// <para>The index memory.</para>
154 /// <para></para>
155 /// </param>
156 [MethodImpl(MethodImplOptions.AggressiveInlining)]
157 protected override void SetPointers(IResizableDirectMemory dataMemory,
        ↳ 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
            ↳ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
            ↳ _linksIndexParts);
165     }
166     else
167     {
168         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
169     }
170     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
171     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
172     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
173     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_linksDataParts, _header);
174 }
175
176 /// <summary>
177 /// <para>
178 /// Resets the pointers.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 [MethodImpl(MethodImplOptions.AggressiveInlining)]
183 protected override void ResetPointers()
184 {

```

```

185     base.ResetPointers();
186     _linksDataParts = null;
187     _linksIndexParts = null;
188     _header = null;
189 }
190
191 /// <summary>
192 /// <para>
193 /// Gets the header reference.
194 /// </para>
195 /// <para></para>
196 /// </summary>
197 /// <returns>
198 /// <para>A ref links header of t link</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
203
204 /// <summary>
205 /// <para>
206 /// Gets the link data part reference using the specified link index.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 /// <param name="linkIndex">
211 /// <para>The link index.</para>
212 /// <para></para>
213 /// </param>
214 /// <returns>
215 /// <para>A ref raw link data part of t link</para>
216 /// <para></para>
217 /// </returns>
218 [MethodImpl(MethodImplOptions.AggressiveInlining)]
219 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
220     => ref _linksDataParts[linkIndex];
221
222 /// <summary>
223 /// <para>
224 /// Gets the link index part reference using the specified link index.
225 /// </para>
226 /// <para></para>
227 /// </summary>
228 /// <param name="linkIndex">
229 /// <para>The link index.</para>
230 /// <para></para>
231 /// </param>
232 /// <returns>
233 /// <para>A ref raw link index part of t link</para>
234 /// <para></para>
235 /// </returns>
236 [MethodImpl(MethodImplOptions.AggressiveInlining)]
237 protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
238     linkIndex) => ref _linksIndexParts[linkIndex];
239
240 /// <summary>
241 /// <para>
242 /// Determines whether this instance are equal.
243 /// </para>
244 /// <para></para>
245 /// </summary>
246 /// <param name="first">
247 /// <para>The first.</para>
248 /// <para></para>
249 /// </param>
250 /// <param name="second">
251 /// <para>The second.</para>
252 /// <para></para>
253 /// </param>
254 /// <returns>
255 /// <para>The bool</para>
256 /// <para></para>
257 /// </returns>
258 [MethodImpl(MethodImplOptions.AggressiveInlining)]
259 protected override bool AreEqual(ulong first, ulong second) => first == second;
260
261 /// <summary>
262 /// <para>

```

```

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

```

```

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

```

```

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

```

1.76 ./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      /// </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 UInt64UnusedLinksListMethods(RawLinkDataPart<ulong>* links, LinksHeader<ulong>*
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

```

1.77 ./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> :
    ↳ SizedAndThreadedAVLBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
23  {
24      private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
    ↳ UncheckedConverter<TLink, long>.Default;
25      private static readonly UncheckedConverter<TLink, int> _addressToInt32Converter =
    ↳ UncheckedConverter<TLink, int>.Default;
26      private static readonly UncheckedConverter<bool, TLink> _boolToAddressConverter =
    ↳ UncheckedConverter<bool, TLink>.Default;
27      private static readonly UncheckedConverter<TLink, bool> _addressToBoolConverter =
    ↳ UncheckedConverter<TLink, bool>.Default;
28      private static readonly UncheckedConverter<int, TLink> _int32ToAddressConverter =
    ↳ UncheckedConverter<int, TLink>.Default;
29
30      /// <summary>
31      /// <para>
32      /// The break.
33      /// </para>
34      /// <para></para>
35      /// </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,
    ↳ byte* header)
79      {
80          Links = links;
81          Header = header;
82          Break = constants.Break;
83          Continue = constants.Continue;
84      }
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,
249         ↪ secondLink.Source, secondLink.Target);
250 }
251
252 /// <summary>
253 /// <para>
254 /// Determines whether this instance first is to the right of second.
255 /// </para>
256 /// <para></para>
257 /// </summary>
258 /// <param name="first">
259 /// <para>The first.</para>
260 /// <para></para>
261 /// </param>
262 /// <param name="second">
263 /// <para>The second.</para>
264 /// <para></para>
265 /// </param>
266 /// <returns>
267 /// <para>The bool</para>
268 /// <para></para>
269 /// </returns>
270 [MethodImpl(MethodImplOptions.AggressiveInlining)]
271 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
272 {
273     ref var firstLink = ref GetLinkReference(first);
274     ref var secondLink = ref GetLinkReference(second);
275     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
276         ↪ secondLink.Source, secondLink.Target);
277 }
278
279 /// <summary>
280 /// <para>
281 /// Gets the size value using the specified value.
282 /// </para>
283 /// <para></para>
284 /// </summary>
285 /// <param name="value">
286 /// <para>The value.</para>
287 /// <para></para>
288 /// </param>
289 /// <returns>
290 /// <para>The link</para>
291 /// <para></para>
292 /// </returns>
293 [MethodImpl(MethodImplOptions.AggressiveInlining)]
294 protected virtual TLink GetSizeValue(TLink value) => Bit<TLink>.PartialRead(value, 5,
295     ↪ -5);
296
297 /// <summary>
298 /// <para>
299 /// Sets the size value using the specified stored value.
300 /// </para>
301 /// <para></para>
302 /// </summary>
303 /// <param name="storedValue">
304 /// <para>The stored value.</para>
305 /// <para></para>
306 /// </param>
307 /// <param name="size">
308 /// <para>The size.</para>
309 /// <para></para>
310 /// </param>
311 [MethodImpl(MethodImplOptions.AggressiveInlining)]
312 protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
    ↪ Bit<TLink>.PartialWrite(storedValue, size, 5, -5);
313
314 /// <summary>
315 /// <para>

```

```

313     /// Determines whether this instance get left is child 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 bool</para>
323     /// <para></para>
324     /// </returns>
325     [MethodImpl(MethodImplOptions.AggressiveInlining)]
326     protected virtual bool GetLeftIsChildValue(TLink value)
327     {
328         unchecked
329         {
330             return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 4, 1));
331             //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 4, 1), default);
332         }
333     }
334
335     /// <summary>
336     /// <para>
337     /// 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
361     /// <summary>
362     /// <para>
363     /// Determines whether this instance get right is child value.
364     /// </para>
365     /// <para></para>
366     /// </summary>
367     /// <param name="value">
368     /// <para>The value.</para>
369     /// <para></para>
370     /// </param>
371     /// <returns>
372     /// <para>The bool</para>
373     /// <para></para>
374     /// </returns>
375     [MethodImpl(MethodImplOptions.AggressiveInlining)]
376     protected virtual bool GetRightIsChildValue(TLink value)
377     {
378         unchecked
379         {
380             return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 3, 1));
381             //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 3, 1), default);
382         }
383     }
384
385     /// <summary>
386     /// <para>
387     /// Sets the right is child value using the specified stored value.
388     /// </para>
389     /// <para></para>
390     /// </summary>

```

```

390     /// <param name="storedValue">
391     /// <para>The stored value.</para>
392     /// <para></para>
393     /// </param>
394     /// <param name="value">
395     /// <para>The value.</para>
396     /// <para></para>
397     /// </param>
398     [MethodImpl(MethodImplOptions.AggressiveInlining)]
399     protected virtual void SetRightIsChildValue(ref TLink storedValue, bool value)
400     {
401         unchecked
402         {
403             var previousValue = storedValue;
404             var modified = Bit<TLink>.PartialWrite(previousValue,
405                 ↪ _boolToAddressConverter.Convert(value), 3, 1);
406             storedValue = modified;
407         }
408     }
409     /// <summary>
410     /// <para>
411     /// Determines whether this instance is child.
412     /// </para>
413     /// <para></para>
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     /// <summary>
435     /// <para>
436     /// Gets the balance value using the specified stored value.
437     /// </para>
438     /// <para></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     /// <summary>
462     /// <para>
463     /// Sets the balance value using the specified stored value.
464     /// </para>
465     /// <para></para>

```

```

465     /// </summary>
466     /// <param name="storedValue">
467     /// <para>The stored value.</para>
468     /// <para></para>
469     /// </param>
470     /// <param name="value">
471     /// <para>The value.</para>
472     /// <para></para>
473     /// </param>
474     [MethodImpl(MethodImplOptions.AggressiveInlining)]
475     protected virtual void SetBalanceValue(ref TLink storedValue, sbyte value)
476     {
477         unchecked
478         {
479             var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
480                 ↪ value & 3);
481             var modified = Bit<TLink>.PartialWrite(storedValue, packagedValue, 0, 3);
482             storedValue = modified;
483         }
484     }
485     /// <summary>
486     /// <para>
487     /// The zero.
488     /// </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
518                 ↪ broken)
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
543             {
544                 root = GetRightOrDefault(root);
545             }
546         }
547     }

```

```

539     }
540     else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
541         ↪ node.Key > root.Key
542     {
543         root = GetRightOrDefault(root);
544     }
545     else // node.Key == root.Key
546     {
547         return root;
548     }
549     return Zero;
550 }
551
552 // TODO: Return indices range instead of references count
553 /// <summary>
554 /// <para>
555 /// Counts the usages using the specified link.
556 /// </para>
557 /// <para></para>
558 /// </summary>
559 /// <param name="link">
560 /// <para>The link.</para>
561 /// <para></para>
562 /// </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 /// <summary>
605 /// <para>
606 /// Eaches the usage using the specified link.
607 /// </para>
608 /// <para></para>
609 /// </summary>
610 /// <param name="link">
611 /// <para>The link.</para>
612 /// <para></para>
613 /// </param>
614 /// <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         /// <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         /// <para></para>
56         /// </summary>
57         protected readonly byte* Header;
58
59         /// <summary>
60         /// <para>
61         /// Initializes a new <see cref="LinksRecursionlessSizeBalancedTreeMethodsBase" />
62         /// ↳ instance.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         /// <param name="constants">
67         /// <para>A constants.</para>
68         /// <para></para>
69         /// </param>
70         /// <param name="links">
71         /// <para>A links.</para>
72         /// <para></para>
73         /// </param>
74         /// <param name="header">
75         /// <para>A header.</para>
76         /// <para></para>
77         /// </param>
78         [MethodImpl(MethodImplOptions.AggressiveInlining)]
79         protected LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
80             ↳ byte* links, byte* header)
81         {

```



```

75     Links = links;
76     Header = header;
77     Break = constants.Break;
78     Continue = constants.Continue;
79 }
80
81 /// <summary>
82 /// <para>
83 /// Gets the tree root.
84 /// </para>
85 /// <para></para>
86 /// </summary>
87 /// <returns>
88 /// <para>The link</para>
89 /// <para></para>
90 /// </returns>
91 [MethodImpl(MethodImplOptions.AggressiveInlining)]
92 protected abstract TLink GetTreeRoot();
93
94 /// <summary>
95 /// <para>
96 /// Gets the base part value using the specified link.
97 /// </para>
98 /// <para></para>
99 /// </summary>
100 /// <param name="link">
101 /// <para>The link.</para>
102 /// <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    /// <para>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    /// <para>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    /// <para>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    /// <para>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 FirstIsToTheLeftOfSecond(TLink first, TLink second)
240    {
241        ref var firstLink = ref GetLinkReference(first);
242        ref var secondLink = ref GetLinkReference(second);
243        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
244            ↪ secondLink.Source, secondLink.Target);
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
    ↪ broken)
305 }
306 }
307
308 /// <summary>
309 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
    ↪ (концом).
310 /// </summary>
311 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
312 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
313 /// <returns>Индекс искомой связи.</returns>
314 [MethodImpl(MethodImplOptions.AggressiveInlining)]
315 public TLink Search(TLink source, TLink target)
316 {
317     var root = GetTreeRoot();
318     while (!EqualToZero(root))
319     {
320         ref var rootLink = ref GetLinkReference(root);
321         var rootSource = rootLink.Source;
322         var rootTarget = rootLink.Target;
323         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
            ↪ node.Key < root.Key
324         {
325             root = GetLeftOrDefault(root);
326         }
327         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
            ↪ node.Key > root.Key
328         {
329             root = GetRightOrDefault(root);
330         }
331         else // node.Key == root.Key
332         {
333             return root;
334         }
335     }
336     return Zero;
337 }
338
339 // TODO: Return indices range instead of references count
340 /// <summary>
341 /// <para>
342 /// Counts the usages using the specified link.
343 /// </para>
344 /// <para></para>
345 /// </summary>
346 /// <param name="link">
347 /// <para>The link.</para>
348 /// <para></para>
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 [MethodImpl(MethodImplOptions.AggressiveInlining)]
416 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
417 {
418     var @continue = Continue;
419     if (EqualToZero(link))
420     {
421         return @continue;
422     }
423     var linkBasePart = GetBasePartValue(link);
424     var @break = Break;
425     if (GreaterThan(linkBasePart, @base))
426     {
427         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
428         {
429             return @break;
430         }
431     }
432     else if (LessThan(linkBasePart, @base))
433     {
434         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
435         {
436             return @break;
437         }
438     }
439     else //if (linkBasePart == @base)
440     {
441         if (AreEqual(handler(GetLinkValues(link)), @break))
442         {
443             return @break;
444         }
445         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
446         {
447             return @break;
448         }
449         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
450         {
451             return @break;
452         }
453     }
454 }

```

```

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

```

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

```

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

```

```

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="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>

```

```

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,
    ↪ secondLink.Source, secondLink.Target);

```

```

270 }
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
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>

```

```

344    /// <para></para>
345    /// </summary>
346    /// <param name="link">
347    /// <para>The link.</para>
348    /// <para></para>
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    [MethodImpl(MethodImplOptions.AggressiveInlining)]
416    private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
417    {
418        var @continue = Continue;
419        if (EqualToZero(link))
420        {
421            return @continue;
422        }

```

```

420     }
421     var linkBasePart = GetBasePartValue(link);
422     var @break = Break;
423     if (GreaterThan(linkBasePart, @base))
424     {
425         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
426         {
427             return @break;
428         }
429     }
430     else if (LessThan(linkBasePart, @base))
431     {
432         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
433         {
434             return @break;
435         }
436     }
437     else //if (linkBasePart == @base)
438     {
439         if (AreEqual(handler(GetLinkValues(link)), @break))
440         {
441             return @break;
442         }
443         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
444         {
445             return @break;
446         }
447         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
448         {
449             return @break;
450         }
451     }
452     return @continue;
453 }
454
455 /// <summary>
456 /// <para>
457 /// Prints the node value using the specified node.
458 /// </para>
459 /// <para></para>
460 /// </summary>
461 /// <param name="node">
462 /// <para>The node.</para>
463 /// <para></para>
464 /// </param>
465 /// <param name="sb">
466 /// <para>The sb.</para>
467 /// <para></para>
468 /// </param>
469 [MethodImpl(MethodImplOptions.AggressiveInlining)]
470 protected override void PrintNodeValue(TLink node, StringBuilder sb)
471 {
472     ref var link = ref GetLinkReference(node);
473     sb.Append(' ');
474     sb.Append(link.Source);
475     sb.Append('-');
476     sb.Append('>');
477     sb.Append(link.Target);
478 }
479 }
480 }

```

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    {

```

```

16    /// <summary>
17    /// <para>
18    /// Initializes a new <see cref="LinksSourcesAvlBalancedTreeMethods"/> 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 LinksSourcesAvlBalancedTreeMethods(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).LeftAsSource;
55
56    /// <summary>
57    /// <para>
58    /// Gets the right reference using the specified node.
59    /// </para>
60    /// <para></para>
61    /// </summary>
62    /// <param name="node">
63    /// <para>The node.</para>
64    /// <para></para>
65    /// </param>
66    /// <returns>
67    /// <para>The ref link</para>
68    /// <para></para>
69    /// </returns>
70    [MethodImpl(MethodImplOptions.AggressiveInlining)]
71    protected override ref TLink GetRightReference(TLink node) => ref
72    ↪ GetLinkReference(node).RightAsSource;
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).LeftAsSource;
90
91    /// <summary>
92    /// <para>
93    /// 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) =>
157   ↪ GetSizeValue(GetLinkReference(node).SizeAsSource);
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>

```

```

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) => SetSizeValue(ref
    ↪ GetLinkReference(node).SizeAsSource, size);

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

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

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

223
224     /// <summary>
225     /// <para>
226     /// Sets the right is child using the specified node.
227     /// </para>
228     /// <para></para>
229     /// </summary>
230     /// <param name="node">
231     /// <para>The node.</para>
232     /// <para></para>
233     /// </param>
234     /// <param name="value">
235     /// <para>The value.</para>
236     /// <para></para>
237     /// </param>
238     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```



```

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,
        ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
        ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));

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

362     /// <summary>
363     /// <para>
364     /// Clears the node using the specified node.
365     /// </para>
366     /// <para></para>
367     /// </summary>
368     /// <param name="node">
369     /// <para>The node.</para>
370     /// <para></para>
371     /// </param>
372     [MethodImpl(MethodImplOptions.AggressiveInlining)]
373     protected override void ClearNode(TLink node)
374     {
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/LinksSourcesRecursionlessSizeBalancedTreeMeth

1 using System.Runtime.CompilerServices;

2

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

```

4
5 namespace Platform.Data.Doublets.Memory.United.Generic
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{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">
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).RightAsSource;
75
76        /// <summary>
77        /// <para>
78        /// Gets the left using the specified node.
79        /// </para>
80        /// <para></para>
81        /// </summary>

```

```

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

```

```

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></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).SizeAsSource = size;
173
174 /// <summary>
175 /// <para>
176 /// Gets the tree root.
177 /// </para>
178 /// <para></para>
179 /// </summary>
180 /// <returns>
181 /// <para>The link</para>
182 /// <para></para>
183 /// </returns>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 protected override 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></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).Source;
203
204 /// <summary>
205 /// <para>
206 /// Determines whether this instance first is to the left of second.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 /// <param name="firstSource">
211 /// <para>The first source.</para>
212 /// <para></para>
213 /// </param>
214 /// <param name="firstTarget">
215 /// <para>The first target.</para>
216 /// <para></para>
217 /// </param>
218 /// <param name="secondSource">
219 /// <para>The second source.</para>
220 /// <para></para>
221 /// </param>
222 /// <param name="secondTarget">
223 /// <para>The second target.</para>
224 /// <para></para>
225 /// </param>
226 /// <returns>
227 /// <para>The bool</para>
228 /// <para></para>
229 /// </returns>
230 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

230     protected override bool FirstIsToLeftOfSecond(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 }

```

1.82 ./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> :
    ↪     LinksSizeBalancedTreeMethodsBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="LinksSourcesSizeBalancedTreeMethods" /> instance.
19         /// </para>
20         /// <para></para>

```

```

21     /// </summary>
22     /// <param name="constants">
23     /// <para>A constants.</para>
24     /// <para></para>
25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public LinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
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).LeftAsSource;
55
56     /// <summary>
57     /// <para>
58     /// Gets the right reference using the specified node.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="node">
63     /// <para>The node.</para>
64     /// <para></para>
65     /// </param>
66     /// <returns>
67     /// <para>The ref link</para>
68     /// <para></para>
69     /// </returns>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     protected override ref TLink GetRightReference(TLink node) => ref
72     ↳ GetLinkReference(node).RightAsSource;
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).LeftAsSource;
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).RightAsSource;

```

```

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

```

```

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

```



```

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 LinksTargetsAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
            ↪ byte* header) : base(constants, links, header) { }
36
37         /// <summary>

```

```

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

```

```

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)]
174     protected override void SetSize(TLink node, TLink size) => SetSizeValue(ref
175         ↪ GetLinkReference(node).SizeAsTarget, size);
176
177     /// <summary>
178     /// <para>
179     /// Determines whether this instance get left is child.
180     /// </para>
181     /// <para></para>
182     /// </summary>
183     /// <param name="node">
184     /// <para>The node.</para>
185     /// <para></para>
186     /// </param>
187     /// <returns>
188     /// <para>The bool</para>
189     /// <para></para>
190     /// </returns>
191     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

188     protected override bool GetLeftIsChild(TLink node) =>
189         ↳ GetLeftIsChildValue(GetLinkReference(node).SizeAsTarget);
190
191     /// <summary>
192     /// <para>
193     /// Sets the left is child using the specified node.
194     /// </para>
195     /// </summary>
196     /// <param name="node">
197     /// <para>The node.</para>
198     /// <para></para>
199     /// </param>
200     /// <param name="value">
201     /// <para>The value.</para>
202     /// <para></para>
203     /// </param>
204     [MethodImpl(MethodImplOptions.AggressiveInlining)]
205     protected override void SetLeftIsChild(TLink node, bool value) =>
206         ↳ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance get right is child.
211     /// </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) =>
223         ↳ GetRightIsChildValue(GetLinkReference(node).SizeAsTarget);
224
225     /// <summary>
226     /// <para>
227     /// Sets the right is child using the specified node.
228     /// </para>
229     /// </summary>
230     /// <param name="node">
231     /// <para>The node.</para>
232     /// <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).SizeAsTarget, value);
241
242     /// <summary>
243     /// <para>
244     /// Gets the balance using the specified node.
245     /// </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>

```

```

260     /// Sets the balance using the specified node.
261     /// </para>
262     /// <para></para>
263     /// </summary>
264     /// <param name="node">
265     /// <para>The node.</para>
266     /// <para></para>
267     /// </param>
268     /// <param name="value">
269     /// <para>The value.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
        ↪ GetLinkReference(node).SizeAsTarget, value);

274
275     /// <summary>
276     /// <para>
277     /// Gets the tree root.
278     /// </para>
279     /// <para></para>
280     /// </summary>
281     /// <returns>
282     /// <para>The link</para>
283     /// <para></para>
284     /// </returns>
285     [MethodImpl(MethodImplOptions.AggressiveInlining)]
286     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;

287
288     /// <summary>
289     /// <para>
290     /// Gets the base part value using the specified link.
291     /// </para>
292     /// <para></para>
293     /// </summary>
294     /// <param name="link">
295     /// <para>The link.</para>
296     /// <para></para>
297     /// </param>
298     /// <returns>
299     /// <para>The link</para>
300     /// <para></para>
301     /// </returns>
302     [MethodImpl(MethodImplOptions.AggressiveInlining)]
303     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Target;

304
305     /// <summary>
306     /// <para>
307     /// Determines whether this instance first is to the left of second.
308     /// </para>
309     /// <para></para>
310     /// </summary>
311     /// <param name="firstSource">
312     /// <para>The first source.</para>
313     /// <para></para>
314     /// </param>
315     /// <param name="firstTarget">
316     /// <para>The first target.</para>
317     /// <para></para>
318     /// </param>
319     /// <param name="secondSource">
320     /// <para>The second source.</para>
321     /// <para></para>
322     /// </param>
323     /// <param name="secondTarget">
324     /// <para>The second target.</para>
325     /// <para></para>
326     /// </param>
327     /// <returns>
328     /// <para>The bool</para>
329     /// <para></para>
330     /// </returns>
331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
332     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
        ↪ 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     /// <summary>
363     /// <para>
364     /// Clears the node using the specified node.
365     /// </para>
366     /// <para></para>
367     /// </summary>
368     /// <param name="node">
369     /// <para>The node.</para>
370     /// <para></para>
371     /// </param>
372     [MethodImpl(MethodImplOptions.AggressiveInlining)]
373     protected override void ClearNode(TLink node)
374     {
375         ref var link = ref GetLinkReference(node);
376         link.LeftAsTarget = Zero;
377         link.RightAsTarget = Zero;
378         link.SizeAsTarget = Zero;
379     }
380 }
381 }
382 }

```

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

```

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

```

```

25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
36         ↳ byte* links, byte* header) : base(constants, links, header) { }
37
38     /// <summary>
39     /// <para>
40     /// Gets the left reference using the specified node.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     /// <param name="node">
45     /// <para>The node.</para>
46     /// <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>

```

```

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
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     /// 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">

```

```

251     /// <para>The second target.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The bool</para>
256     /// <para></para>
257     /// </returns>
258     [MethodImpl(MethodImplOptions.AggressiveInlining)]
259     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));

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

```

1.85 ./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,
            ↪ byte* header) : base(constants, links, header) { }
36
37         /// <summary>
38         /// <para>
39         /// Gets the left reference using the specified node.
40         /// </para>
41         /// <para></para>
42         /// </summary>

```

```

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

```

```

119 [MethodImpl(MethodImplOptions.AggressiveInlining)]
120 protected override void SetLeft(TLink node, TLink left) =>
    ↳ GetLinkReference(node).LeftAsTarget = left;
121
122 /// <summary>
123 /// <para>
124 /// Sets the right using the specified node.
125 /// </para>
126 /// <para></para>
127 /// </summary>
128 /// <param name="node">
129 /// <para>The node.</para>
130 /// <para></para>
131 /// </param>
132 /// <param name="right">
133 /// <para>The right.</para>
134 /// <para></para>
135 /// </param>
136 [MethodImpl(MethodImplOptions.AggressiveInlining)]
137 protected override void SetRight(TLink node, TLink right) =>
    ↳ GetLinkReference(node).RightAsTarget = right;
138
139 /// <summary>
140 /// <para>
141 /// Gets the size using the specified node.
142 /// </para>
143 /// <para></para>
144 /// </summary>
145 /// <param name="node">
146 /// <para>The node.</para>
147 /// <para></para>
148 /// </param>
149 /// <returns>
150 /// <para>The link</para>
151 /// <para></para>
152 /// </returns>
153 [MethodImpl(MethodImplOptions.AggressiveInlining)]
154 protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsTarget;
155
156 /// <summary>
157 /// <para>
158 /// Sets the size using the specified node.
159 /// </para>
160 /// <para></para>
161 /// </summary>
162 /// <param name="node">
163 /// <para>The node.</para>
164 /// <para></para>
165 /// </param>
166 /// <param name="size">
167 /// <para>The size.</para>
168 /// <para></para>
169 /// </param>
170 [MethodImpl(MethodImplOptions.AggressiveInlining)]
171 protected override void SetSize(TLink node, TLink size) =>
    ↳ GetLinkReference(node).SizeAsTarget = 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().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>
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(TLink firstSource, TLink firstTarget,
262     ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
263     ↪ (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>

```

```

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)]
46         public UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
47         ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
48         ↪ memoryReservationStep) { }
49
50         /// <summary>
51         /// <para>
52         /// Initializes a new <see cref="UnitedMemoryLinks" /> instance.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <param name="memory">
57         /// <para>A memory.</para>
58         /// <para></para>
59         /// </param>
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         public UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
62         ↪ DefaultLinksSizeStep) { }

```

```

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

```

```

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     /// </param>
171     /// <returns>
172     /// <para>A ref raw link of t link</para>
173     /// <para></para>
174     /// </returns>
175     [MethodImpl(MethodImplOptions.AggressiveInlining)]
176     protected override ref RawLink<TLink> GetLinkReference(TLink linkIndex) => ref
177         ↪ AsRef<RawLink<TLink>>(_links + (LinkSizeInBytes * Convert.ToInt64(linkIndex)));
178 }

```

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 =
    ↳ EqualityComparer<TLink>.Default;
26 private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
27 private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
    ↳ UncheckedConverter<TLink, long>.Default;
28 private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
    ↳ UncheckedConverter<long, TLink>.Default;
29 private static readonly TLink _zero = default;
30 private static readonly TLink _one = Arithmetic.Increment(_zero);
31
32 /// <summary>Возвращает размер одной связи в байтах.</summary>
33 /// <remarks>
34 /// Используется только во вне класса, не рекомендуется использовать внутри.
35 /// Так как во вне не обязательно будет доступен unsafe C#.
36 /// </remarks>
37 public static readonly long LinkSizeInBytes = RawLink<TLink>.SizeInBytes;
38
39 /// <summary>
40 /// <para>
41 /// The size in bytes.
42 /// </para>
43 /// <para></para>
44 /// </summary>
45 public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
46
47 /// <summary>
48 /// <para>
49 /// The link size in bytes.
50 /// </para>
51 /// <para></para>
52 /// </summary>
53 public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
54
55 /// <summary>
56 /// <para>
57 /// The memory.
58 /// </para>
59 /// <para></para>
60 /// </summary>
61 protected readonly IResizableDirectMemory _memory;
62 /// <summary>
63 /// <para>
64 /// The memory reservation step.
65 /// </para>
66 /// <para></para>
67 /// </summary>
68 protected readonly long _memoryReservationStep;
69
70 /// <summary>
71 /// <para>
72 /// The targets tree methods.
73 /// </para>
74 /// <para></para>
75 /// </summary>
76 protected ILinksTreeMethods<TLink> TargetsTreeMethods;
77 /// <summary>
78 /// <para>
79 /// The sources tree methods.
80 /// </para>
81 /// <para></para>
82 /// </summary>
83 protected ILinksTreeMethods<TLink> SourcesTreeMethods;
84 // 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
85 /// <summary>
86 /// <para>
87 /// The unused links list methods.
88 /// </para>
89 /// <para></para>
90 /// </summary>
91 protected ILinksListMethods<TLink> UnusedLinksListMethods;
92
93 /// <summary>
94 /// Возвращает общее число связей находящихся в хранилище.
95 /// </summary>
96 protected virtual TLink Total
97 {
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

99         get
100     {
101         ref var header = ref GetHeaderReference();
102         return Subtract(header.AllocatedLinks, header.FreeLinks);
103     }
104 }
105
106 /// <summary>
107 /// <para>
108 /// Gets the constants value.
109 /// </para>
110 /// <para></para>
111 /// </summary>
112 public virtual LinksConstants<TLink> Constants
113 {
114     [MethodImpl(MethodImplOptions.AggressiveInlining)]
115     get;
116 }
117
118 /// <summary>
119 /// <para>
120 /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
121 /// </para>
122 /// <para></para>
123 /// </summary>
124 /// <param name="memory">
125 /// <para>A memory.</para>
126 /// <para></para>
127 /// </param>
128 /// <param name="memoryReservationStep">
129 /// <para>A memory reservation step.</para>
130 /// <para></para>
131 /// </param>
132 /// <param name="constants">
133 /// <para>A constants.</para>
134 /// <para></para>
135 /// </param>
136 [MethodImpl(MethodImplOptions.AggressiveInlining)]
137 protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
    ↪ memoryReservationStep, LinksConstants<TLink> constants)
138 {
139     _memory = memory;
140     _memoryReservationStep = memoryReservationStep;
141     Constants = constants;
142 }
143
144 /// <summary>
145 /// <para>
146 /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
147 /// </para>
148 /// <para></para>
149 /// </summary>
150 /// <param name="memory">
151 /// <para>A memory.</para>
152 /// <para></para>
153 /// </param>
154 /// <param name="memoryReservationStep">
155 /// <para>A memory reservation step.</para>
156 /// <para></para>
157 /// </param>
158 [MethodImpl(MethodImplOptions.AggressiveInlining)]
159 protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
    ↪ memoryReservationStep) : this(memory, memoryReservationStep,
    ↪ Default<LinksConstants<TLink>>.Instance) { }
160
161 /// <summary>
162 /// <para>
163 /// Inits the memory.
164 /// </para>
165 /// <para></para>
166 /// </summary>
167 /// <param name="memory">
168 /// <para>The memory.</para>
169 /// <para></para>
170 /// </param>
171 /// <param name="memoryReservationStep">
172 /// <para>The memory reservation step.</para>
173 /// <para></para>

```

```

174 /// </param>
175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
176 protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
177 {
178     if (memory.ReservedCapacity < memoryReservationStep)
179     {
180         memory.ReservedCapacity = memoryReservationStep;
181     }
182     SetPointers(memory);
183     ref var header = ref GetHeaderReference();
184     // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
185     memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
186         ↪ LinkHeaderSizeInBytes;
187     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
188     header.ReservedLinks = ConvertToAddress((memory.ReservedCapacity -
189         ↪ LinkHeaderSizeInBytes) / LinkSizeInBytes);
189 }
190
191 /// <summary>
192 /// <para>
193 /// Counts the restrictions.
194 /// </para>
195 /// </summary>
196 /// <param name="restrictions">
197 /// <para>The restrictions.</para>
198 /// </para>
199 /// </param>
200 /// <exception cref="NotSupportedException">
201 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
202 /// </exception>
203 /// </returns>
204 /// <para>The link</para>
205 /// </returns>
206 [MethodImpl(MethodImplOptions.AggressiveInlining)]
207 public virtual TLink Count(IList<TLink> restrictions)
208 {
209     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
210     if (restrictions.Count == 0)
211     {
212         return Total;
213     }
214     var constants = Constants;
215     var any = constants.Any;
216     var index = restrictions[constants.IndexPart];
217     if (restrictions.Count == 1)
218     {
219         if (AreEqual(index, any))
220         {
221             return Total;
222         }
223         return Exists(index) ? GetOne() : GetZero();
224     }
225     if (restrictions.Count == 2)
226     {
227         var value = restrictions[1];
228         if (AreEqual(index, any))
229         {
230             if (AreEqual(value, any))
231             {
232                 return Total; // Any - как отсутствие ограничения
233             }
234             return Add(SourcesTreeMethods.CountUsages(value),
235                 ↪ TargetsTreeMethods.CountUsages(value));
236         }
237         else
238         {
239             if (!Exists(index))
240             {
241                 return GetZero();
242             }
243             if (AreEqual(value, any))
244             {
245                 return GetOne();
246             }
247             ref var storedLinkValue = ref GetLinkReference(index);
248

```

```

249         if (AreEqual(storedLinkValue.Source, value) ||
250             ↪ AreEqual(storedLinkValue.Target, value))
251         {
252             return GetOne();
253         }
254         return GetZero();
255     }
256     if (restrictions.Count == 3)
257     {
258         var source = restrictions[constants.SourcePart];
259         var target = restrictions[constants.TargetPart];
260         if (AreEqual(index, any))
261         {
262             if (AreEqual(source, any) && AreEqual(target, any))
263             {
264                 return Total;
265             }
266             else if (AreEqual(source, any))
267             {
268                 return TargetsTreeMethods.CountUsages(target);
269             }
270             else if (AreEqual(target, any))
271             {
272                 return SourcesTreeMethods.CountUsages(source);
273             }
274             else //if(source != Any && target != Any)
275             {
276                 // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
277                 var link = SourcesTreeMethods.Search(source, target);
278                 return AreEqual(link, constants.Null) ? GetZero() : GetOne();
279             }
280         }
281         else
282         {
283             if (!Exists(index))
284             {
285                 return GetZero();
286             }
287             if (AreEqual(source, any) && AreEqual(target, any))
288             {
289                 return GetOne();
290             }
291             ref var storedLinkValue = ref GetLinkReference(index);
292             if (!AreEqual(source, any) && !AreEqual(target, any))
293             {
294                 if (AreEqual(storedLinkValue.Source, source) &&
295                     ↪ AreEqual(storedLinkValue.Target, target))
296                 {
297                     return GetOne();
298                 }
299                 return GetZero();
300             }
301             var value = default(TLink);
302             if (AreEqual(source, any))
303             {
304                 value = target;
305             }
306             if (AreEqual(target, any))
307             {
308                 value = source;
309             }
310             if (AreEqual(storedLinkValue.Source, value) ||
311                 ↪ AreEqual(storedLinkValue.Target, value))
312             {
313                 return GetOne();
314             }
315             return GetZero();
316         }
317     }
318     throw new NotSupportedException("Другие размеры и способы ограничений не
319     ↪ поддерживаются.");
320 }
321
322 /// <summary>
323 /// <para>
324 /// Eaches the handler.
325 /// </para>

```

```

323 /// <para></para>
324 /// </summary>
325 /// <param name="handler">
326 /// <para>The handler.</para>
327 /// <para></para>
328 /// </param>
329 /// <param name="restrictions">
330 /// <para>The restrictions.</para>
331 /// <para></para>
332 /// </param>
333 /// <exception cref="NotSupportedException">
334 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
335 /// <para></para>
336 /// </exception>
337 /// <returns>
338 /// <para>The link</para>
339 /// <para></para>
340 /// </returns>
341 [MethodImpl(MethodImplOptions.AggressiveInlining)]
342 public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
343 {
344     var constants = Constants;
345     var @break = constants.Break;
346     if (restrictions.Count == 0)
347     {
348         for (var link = GetOne(); LessOrEqualThan(link,
349             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
350         {
351             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
352             {
353                 return @break;
354             }
355             return @break;
356         }
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))

```

```

400         {
401             return handler(GetLinkStruct(index));
402         }
403         return @continue;
404     }
405 }
406 if (restrictions.Count == 3)
407 {
408     var source = restrictions[constants.SourcePart];
409     var target = restrictions[constants.TargetPart];
410     if (AreEqual(index, any))
411     {
412         if (AreEqual(source, any) && AreEqual(target, any))
413         {
414             return Each(handler, Array.Empty<TLink>());
415         }
416         else if (AreEqual(source, any))
417         {
418             return TargetsTreeMethods.EachUsage(target, handler);
419         }
420         else if (AreEqual(target, any))
421         {
422             return SourcesTreeMethods.EachUsage(source, handler);
423         }
424         else //if(source != Any && target != Any)
425         {
426             var link = SourcesTreeMethods.Search(source, target);
427             return AreEqual(link, constants.Null) ? @continue :
428                 ↪ handler(GetLinkStruct(link));
429         }
430     }
431     else
432     {
433         if (!Exists(index))
434         {
435             return @continue;
436         }
437         if (AreEqual(source, any) && AreEqual(target, any))
438         {
439             return handler(GetLinkStruct(index));
440         }
441         ref var storedLinkValue = ref GetLinkReference(index);
442         if (!AreEqual(source, any) && !AreEqual(target, any))
443         {
444             if (AreEqual(storedLinkValue.Source, source) &&
445                 AreEqual(storedLinkValue.Target, target))
446             {
447                 return handler(GetLinkStruct(index));
448             }
449             return @continue;
450         }
451         var value = default(TLink);
452         if (AreEqual(source, any))
453         {
454             value = target;
455         }
456         if (AreEqual(target, any))
457         {
458             value = source;
459         }
460         if (AreEqual(storedLinkValue.Source, value) ||
461             AreEqual(storedLinkValue.Target, value))
462         {
463             return handler(GetLinkStruct(index));
464         }
465         return @continue;
466     }
467 }
468 throw new NotSupportedException("Другие размеры и способы ограничений не
469     ↪ поддерживаются.");
470 }
471
472 /// <remarks>
473 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
474     ↪ в другом месте (но не в менеджере памяти, а в логике Links)
475 /// </remarks>
476 [MethodImpl(MethodImplOptions.AggressiveInlining)]
477 public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)

```

```

475 {
476     var constants = Constants;
477     var @null = constants.Null;
478     var linkIndex = restrictions[constants.IndexPart];
479     ref var link = ref GetLinkReference(linkIndex);
480     ref var header = ref GetHeaderReference();
481     ref var firstAsSource = ref header.RootAsSource;
482     ref var firstAsTarget = ref header.RootAsTarget;
483     // Будет корректно работать только в том случае, если пространство выделенной связи
484     ↪ 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
508 ↪ пространство
509 /// </remarks>
510 [MethodImpl(MethodImplOptions.AggressiveInlining)]
511 public virtual TLink Create(ICollection<TLink> restrictions)
512 {
513     ref var header = ref GetHeaderReference();
514     var freeLink = header.FirstFreeLink;
515     if (!AreEqual(freeLink, Constants.Null))
516     {
517         UnusedLinksListMethods.Detach(freeLink);
518     }
519     else
520     {
521         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
522         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
523         {
524             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
525         }
526         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
527         {
528             _memory.ReservedCapacity += _memory.ReservationStep;
529             SetPointers(_memory);
530             header = ref GetHeaderReference();
531             header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
532             ↪ LinkSizeInBytes);
533         }
534         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
535         _memory.UsedCapacity += LinkSizeInBytes;
536     }
537     return freeLink;
538 }
539
540 /// <summary>
541 /// <para>
542 /// Deletes the restrictions.
543 /// </para>
544 /// </summary>
545 /// <param name="restrictions">
546 /// <para>The restrictions.</para>
547 /// </param>
548 [MethodImpl(MethodImplOptions.AggressiveInlining)]
549 public virtual void Delete(ICollection<TLink> restrictions)
550 {
551     ref var header = ref GetHeaderReference();

```

```

551     var link = restrictions[Constants.IndexPart];
552     if (LessThan(link, header.AllocatedLinks))
553     {
554         UnusedLinksListMethods.AttachAsFirst(link);
555     }
556     else if (AreEqual(link, header.AllocatedLinks))
557     {
558         header.AllocatedLinks = Decrement(header.AllocatedLinks);
559         _memory.UsedCapacity -= LinkSizeInBytes;
560         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
561         // ↳ пока не дойдём до первой существующей связи
562         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
563         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
564             ↳ IsUnusedLink(header.AllocatedLinks))
565         {
566             UnusedLinksListMethods.Detach(header.AllocatedLinks);
567             header.AllocatedLinks = Decrement(header.AllocatedLinks);
568             _memory.UsedCapacity -= LinkSizeInBytes;
569         }
570     }
571     }
572     /// <summary>
573     /// <para>
574     /// Gets the link struct using the specified link index.
575     /// </para>
576     /// <para></para>
577     /// </summary>
578     /// <param name="linkIndex">
579     /// <para>The link index.</para>
580     /// <para></para>
581     /// </param>
582     /// <returns>
583     /// <para>A list of t link</para>
584     /// <para></para>
585     /// </returns>
586     [MethodImpl(MethodImplOptions.AggressiveInlining)]
587     public IList<TLink> GetLinkStruct(TLink linkIndex)
588     {
589         ref var link = ref GetLinkReference(linkIndex);
590         return new Link<TLink>(linkIndex, link.Source, link.Target);
591     }
592     /// <remarks>
593     /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
594     /// ↳ адрес реально поменялся
595     ///
596     /// Указатель this.links может быть в том же месте,
597     /// так как 0-я связь не используется и имеет такой же размер как Header,
598     /// поэтому header размещается в том же месте, что и 0-я связь
599     /// </remarks>
600     [MethodImpl(MethodImplOptions.AggressiveInlining)]
601     protected abstract void SetPointers(IResizableDirectMemory memory);
602     /// <summary>
603     /// <para>
604     /// Resets the pointers.
605     /// </para>
606     /// <para></para>
607     /// </summary>
608     [MethodImpl(MethodImplOptions.AggressiveInlining)]
609     protected virtual void ResetPointers()
610     {
611         SourcesTreeMethods = null;
612         TargetsTreeMethods = null;
613         UnusedLinksListMethods = null;
614     }
615     /// <summary>
616     /// <para>
617     /// Gets the header reference.
618     /// </para>
619     /// <para></para>
620     /// </summary>
621     /// <returns>
622     /// <para>A ref links header of t link</para>
623     /// <para></para>
624     /// </returns>

```



```

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

```

```

703     /// </returns>
704     [MethodImpl(MethodImplOptions.AggressiveInlining)]
705     protected virtual TLink GetOne() => _one;
706
707     /// <summary>
708     /// <para>
709     /// Gets the zero.
710     /// </para>
711     /// <para></para>
712     /// </summary>
713     /// <returns>
714     /// <para>The link</para>
715     /// <para></para>
716     /// </returns>
717     [MethodImpl(MethodImplOptions.AggressiveInlining)]
718     protected virtual TLink GetZero() => default;
719
720     /// <summary>
721     /// <para>
722     /// Determines whether this instance are equal.
723     /// </para>
724     /// <para></para>
725     /// </summary>
726     /// <param name="first">
727     /// <para>The first.</para>
728     /// <para></para>
729     /// </param>
730     /// <param name="second">
731     /// <para>The second.</para>
732     /// <para></para>
733     /// </param>
734     /// <returns>
735     /// <para>The bool</para>
736     /// <para></para>
737     /// </returns>
738     [MethodImpl(MethodImplOptions.AggressiveInlining)]
739     protected virtual bool AreEqual(TLink first, TLink second) =>
740         ↪ _equalityComparer.Equals(first, second);
741
742     /// <summary>
743     /// <para>
744     /// Determines whether this instance less than.
745     /// </para>
746     /// <para></para>
747     /// </summary>
748     /// <param name="first">
749     /// <para>The first.</para>
750     /// <para></para>
751     /// </param>
752     /// <param name="second">
753     /// <para>The second.</para>
754     /// <para></para>
755     /// </param>
756     /// <returns>
757     /// <para>The bool</para>
758     /// <para></para>
759     /// </returns>
760     [MethodImpl(MethodImplOptions.AggressiveInlining)]
761     protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
762         ↪ second) < 0;
763
764     /// <summary>
765     /// <para>
766     /// Determines whether this instance less or equal than.
767     /// </para>
768     /// <para></para>
769     /// </summary>
770     /// <param name="first">
771     /// <para>The first.</para>
772     /// <para></para>
773     /// </param>
774     /// <param name="second">
775     /// <para>The second.</para>
776     /// <para></para>
777     /// </param>
778     /// <returns>
779     /// <para>The bool</para>
780     /// <para></para>

```

```

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

```

```

853     /// <para>The link</para>
854     /// <para></para>
855     /// </returns>
856     [MethodImpl(MethodImplOptions.AggressiveInlining)]
857     protected virtual TLink ConvertToAddress(long value) =>
        ↪ _int64ToAddressConverter.Convert(value);

858
859     /// <summary>
860     /// <para>
861     /// Adds the first.
862     /// </para>
863     /// <para></para>
864     /// </summary>
865     /// <param name="first">
866     /// <para>The first.</para>
867     /// <para></para>
868     /// </param>
869     /// <param name="second">
870     /// <para>The second.</para>
871     /// <para></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,
        ↪ second);

879
880     /// <summary>
881     /// <para>
882     /// Subtracts the first.
883     /// </para>
884     /// <para></para>
885     /// </summary>
886     /// <param name="first">
887     /// <para>The first.</para>
888     /// <para></para>
889     /// </param>
890     /// <param name="second">
891     /// <para>The second.</para>
892     /// <para></para>
893     /// </param>
894     /// <returns>
895     /// <para>The link</para>
896     /// <para></para>
897     /// </returns>
898     [MethodImpl(MethodImplOptions.AggressiveInlining)]
899     protected virtual TLink Subtract(TLink first, TLink second) =>
        ↪ 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     /// <para></para>
910     /// </param>
911     /// <returns>
912     /// <para>The link</para>
913     /// <para></para>
914     /// </returns>
915     [MethodImpl(MethodImplOptions.AggressiveInlining)]
916     protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);

917
918     /// <summary>
919     /// <para>
920     /// Decrements the link.
921     /// </para>
922     /// <para></para>
923     /// </summary>
924     /// <param name="link">
925     /// <para>The link.</para>
926     /// <para></para>
927     /// </param>

```

```

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

```

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> :
19         ↳ AbsoluteCircularDoublyLinkedListMethods<TLink>, ILinksListMethods<TLink>
20     {
21         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
22             ↳ UncheckedConverter<TLink, long>.Default;
23         private readonly byte* _links;
24         private readonly byte* _header;
25
26         /// <summary>
27         /// <para>
28         /// Initializes a new <see cref="UnusedLinksListMethods" /> instance.
29         /// </para>

```

```

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

```

```

103     /// Gets the previous using the specified element.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="element">
108     /// <para>The element.</para>
109     /// <para></para>
110     /// </param>
111     /// <returns>
112     /// <para>The link</para>
113     /// <para></para>
114     /// </returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     protected override TLink GetPrevious(TLink element) => GetLinkReference(element).Source;
117
118     /// <summary>
119     /// <para>
120     /// Gets the next using the specified element.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="element">
125     /// <para>The element.</para>
126     /// <para></para>
127     /// </param>
128     /// <returns>
129     /// <para>The link</para>
130     /// <para></para>
131     /// </returns>
132     [MethodImpl(MethodImplOptions.AggressiveInlining)]
133     protected override TLink GetNext(TLink element) => GetLinkReference(element).Target;
134
135     /// <summary>
136     /// <para>
137     /// Gets the size.
138     /// </para>
139     /// <para></para>
140     /// </summary>
141     /// <returns>
142     /// <para>The link</para>
143     /// <para></para>
144     /// </returns>
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     protected override TLink GetSize() => GetHeaderReference().FreeLinks;
147
148     /// <summary>
149     /// <para>
150     /// Sets the first using the specified element.
151     /// </para>
152     /// <para></para>
153     /// </summary>
154     /// <param name="element">
155     /// <para>The element.</para>
156     /// <para></para>
157     /// </param>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
160         ↪ 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 =
174         ↪ element;
175
176     /// <summary>
177     /// <para>
178     /// Sets the previous using the specified element.
179     /// </para>
180     /// <para></para>
181     /// </summary>
182     /// <param name="element">
183     /// <para>The element.</para>
184     /// <para></para>
185     /// </param>
186     /// <returns>
187     /// <para>The link</para>
188     /// <para></para>
189     /// </returns>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override TLink SetPrevious(TLink element) => GetLinkReference(element).Source;
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     /// <returns>
204     /// <para>The link</para>
205     /// <para></para>
206     /// </returns>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override TLink SetNext(TLink element) => GetLinkReference(element).Target;
209
210     /// <summary>
211     /// <para>
212     /// Gets the size.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     /// <returns>
217     /// <para>The link</para>
218     /// <para></para>
219     /// </returns>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     protected override TLink GetSize() => GetHeaderReference().FreeLinks;
222
223     /// <summary>
224     /// <para>
225     /// Sets the first using the specified element.
226     /// </para>
227     /// <para></para>
228     /// </summary>
229     /// <param name="element">
230     /// <para>The element.</para>
231     /// <para></para>
232     /// </param>
233     [MethodImpl(MethodImplOptions.AggressiveInlining)]
234     protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
235         ↪ element;
236
237     /// <summary>
238     /// <para>
239     /// Sets the last using the specified element.
240     /// </para>
241     /// <para></para>
242     /// </summary>
243     /// <param name="element">
244     /// <para>The element.</para>
245     /// <para></para>
246     /// </param>
247     [MethodImpl(MethodImplOptions.AggressiveInlining)]
248     protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
249         ↪ element;
250
251     /// <summary>
252     /// <para>
253     /// Sets the previous using the specified element.
254     /// </para>
255     /// <para></para>
256     /// </summary>
257     /// <param name="element">
258     /// <para>The element.</para>
259     /// <para></para>
260     /// </param>
261     /// <returns>
262     /// <para>The link</para>
263     /// <para></para>
264     /// </returns>
265     [MethodImpl(MethodImplOptions.AggressiveInlining)]
266     protected override TLink SetPrevious(TLink element) => GetLinkReference(element).Source;
267
268     /// <summary>
269     /// <para>
270     /// Sets the next using the specified element.
271     /// </para>
272     /// <para></para>
273     /// </summary>
274     /// <param name="element">
275     /// <para>The element.</para>
276     /// <para></para>
277     /// </param>
278     /// <returns>
279     /// <para>The link</para>
280     /// <para></para>
281     /// </returns>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override TLink SetNext(TLink element) => GetLinkReference(element).Target;

```

```

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
192     /// <summary>
193     /// <para>
194     /// Sets the next using the specified element.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="element">
199     /// <para>The element.</para>
200     /// <para></para>
201     /// </param>
202     /// <param name="next">
203     /// <para>The next.</para>
204     /// <para></para>
205     /// </param>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override void SetNext(TLink element, TLink next) =>
208         ↪ GetLinkReference(element).Target = next;
209
210     /// <summary>
211     /// <para>
212     /// Sets the size using the specified size.
213     /// </para>
214     /// <para></para>
215     /// </summary>
216     /// <param name="size">
217     /// <para>The size.</para>
218     /// <para></para>
219     /// </param>
220     [MethodImpl(MethodImplOptions.AggressiveInlining)]
221     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
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         /// <para></para>
26         /// </summary>
27         public static readonly long SizeInBytes = Structure<RawLink<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     /// <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,
139     ↪ SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
140
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     public static bool operator ==(RawLink<TLink> left, RawLink<TLink> right) =>
143     ↪ left.Equals(right);
144
145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static bool operator !=(RawLink<TLink> left, RawLink<TLink> right) => !(left ==
147     ↪ right);
148 }
149 }

```

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 :
16     ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<uint>
17     {
18         /// <summary>
19         /// <para>
20         /// The links.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         protected new readonly RawLink<uint>* Links;
25
26         /// <summary>
27         /// <para>
28         /// The header.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         protected new readonly LinksHeader<uint>* Header;
33
34         /// <summary>
35         /// <para>
36         /// Initializes a new <see cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
37         ↪ instance.

```

```

35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="constants">
39     /// <para>A constants.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="links">
43     /// <para>A links.</para>
44     /// <para></para>
45     /// </param>
46     /// <param name="header">
47     /// <para>A header.</para>
48     /// <para></para>
49     /// </param>
50     protected UInt32LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<uint>
    ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header)
51         : base(constants, (byte*)links, (byte*)header)
52     {
53         Links = links;
54         Header = header;
55     }
56
57     /// <summary>
58     /// <para>
59     /// Gets the zero.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <returns>
64     /// <para>The uint</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override uint GetZero() => 0U;
69
70     /// <summary>
71     /// <para>
72     /// Determines whether this instance equal to zero.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="value">
77     /// <para>The value.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The bool</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override bool EqualToZero(uint value) => value == 0U;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance are equal.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="first">
94     /// <para>The first.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="second">
98     /// <para>The second.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The bool</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override bool AreEqual(uint first, uint second) => first == second;
107
108    /// <summary>
109    /// <para>
110    /// Determines whether this instance greater than zero.
111    /// </para>

```

```

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

```

```

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

```

```

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

```

```

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

```

1.91 ./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        /// <summary>
26        /// <para>
27        /// The header.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        protected new readonly LinksHeader<uint>* Header;
32
33        /// <summary>
34        /// <para>
35        /// Initializes a new <see cref="UInt32LinksSizeBalancedTreeMethodsBase"/> 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 UInt32LinksSizeBalancedTreeMethodsBase(LinksConstants<uint> constants,
52    ↪ 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>
```



```

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>
289     /// <para>The uint</para>
290     /// <para></para>
291     /// </returns>
292     [MethodImpl(MethodImplOptions.AggressiveInlining)]
293     protected override uint Decrement(uint value) => --value;
294
295     /// <summary>
296     /// <para>
297     /// Adds the first.
298     /// </para>
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>

```

```

307     /// </param>
308     /// <returns>
309     /// <para>The uint</para>
310     /// <para></para>
311     /// </returns>
312     [MethodImpl(MethodImplOptions.AggressiveInlining)]
313     protected override uint Add(uint first, uint second) => first + second;
314
315     /// <summary>
316     /// <para>
317     /// Subtracts the first.
318     /// </para>
319     /// <para></para>
320     /// </summary>
321     /// <param name="first">
322     /// <para>The first.</para>
323     /// <para></para>
324     /// </param>
325     /// <param name="second">
326     /// <para>The second.</para>
327     /// <para></para>
328     /// </param>
329     /// <returns>
330     /// <para>The uint</para>
331     /// <para></para>
332     /// </returns>
333     [MethodImpl(MethodImplOptions.AggressiveInlining)]
334     protected override uint Subtract(uint first, uint second) => first - second;
335
336     /// <summary>
337     /// <para>
338     /// Determines whether this instance first is to the left of second.
339     /// </para>
340     /// <para></para>
341     /// </summary>
342     /// <param name="first">
343     /// <para>The first.</para>
344     /// <para></para>
345     /// </param>
346     /// <param name="second">
347     /// <para>The second.</para>
348     /// <para></para>
349     /// </param>
350     /// <returns>
351     /// <para>The bool</para>
352     /// <para></para>
353     /// </returns>
354     [MethodImpl(MethodImplOptions.AggressiveInlining)]
355     protected override bool FirstIsToLeftOfSecond(uint first, uint second)
356     {
357         ref var firstLink = ref Links[first];
358         ref var secondLink = ref Links[second];
359         return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
360             ↪ secondLink.Source, secondLink.Target);
361     }
362
363     /// <summary>
364     /// <para>
365     /// Determines whether this instance first is to the right of second.
366     /// </para>
367     /// <para></para>
368     /// </summary>
369     /// <param name="first">
370     /// <para>The first.</para>
371     /// <para></para>
372     /// </param>
373     /// <param name="second">
374     /// <para>The second.</para>
375     /// <para></para>
376     /// </param>
377     /// <returns>
378     /// <para>The bool</para>
379     /// <para></para>
380     /// </returns>
381     [MethodImpl(MethodImplOptions.AggressiveInlining)]
382     protected override bool FirstIsToTheRightOfSecond(uint first, uint second)
383     {
384         ref var firstLink = ref Links[first];

```

```

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

```

1.92 ./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 :
        ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see
        ↪ cref="UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 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
69     ↪ Links[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 uint</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override uint GetLeft(uint node) => Links[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 uint</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    protected override uint GetRight(uint node) => Links[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>

```

```

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

```

```

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

```



```

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 :
15         ↳ UInt32LinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32LinksSourcesSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="links">
28         /// <para>A links.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="header">
32         /// <para>A header.</para>
33         /// <para></para>
34         /// </param>
35         public UInt32LinksSourcesSizeBalancedTreeMethods(LinksConstants<uint> constants,
36             ↳ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
37
38         /// <summary>
39         /// <para>
40         /// Gets the left reference using the specified node.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         /// <param name="node">
45         /// <para>The node.</para>
46         /// <para></para>
47         /// </param>
48         /// <returns>
49         /// <para>The ref uint</para>
50         /// <para></para>
51         /// </returns>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
54
55         /// <summary>
56         /// <para>
57         /// Gets the right reference using the specified node.
58         /// </para>
59         /// </summary>
60         /// <param name="node">
61         /// <para>The node.</para>
62         /// <para></para>
63         /// </param>
64         /// <returns>
65         /// <para>The ref uint</para>
66         /// <para></para>
67         /// </returns>
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         protected override ref uint GetRightReference(uint node) => ref Links[node].RightAsSource;
70     }
71 }

```

```

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

```

```

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

```

```

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

```

1.94 ./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
73             ↳ Links[node].RightAsTarget;
74
75         /// <summary>
76         /// <para>
77         /// Gets the left using the specified node.
78         /// </para>
79         /// </summary>
80         /// <param name="node">
81         /// <para>The node.</para>
82         /// <para></para>
83         /// </param>
84         /// <returns>
85         /// <para>The ref uint</para>
86         /// <para></para>
87         /// </returns>
88         [MethodImpl(MethodImplOptions.AggressiveInlining)]
89         protected override ref uint GetLeftUsingNode(uint node) => ref
90             ↳ Links[node].LeftUsingNode;
91
92         /// <summary>
93         /// <para>
94         /// Gets the right using the specified node.
95         /// </para>
96         /// </summary>
97         /// <param name="node">
98         /// <para>The node.</para>
99         /// <para></para>
100        /// </param>
101        /// <returns>
102        /// <para>The ref uint</para>
103        /// <para></para>
104        /// </returns>
105        [MethodImpl(MethodImplOptions.AggressiveInlining)]
106        protected override ref uint GetRightUsingNode(uint node) => ref
107            ↳ Links[node].RightUsingNode;
108    }
109 }
```

```

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 :
    ↪ UInt32LinksSizeBalancedTreeMethodsBase
15    {
16        /// <summary>
17        /// <para>

```



```

18     /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> instance.
19     /// </para>
20     /// </summary>
21     /// <param name="constants">
22     /// <para>A constants.</para>
23     /// </param>
24     /// <param name="links">
25     /// <para>A links.</para>
26     /// </param>
27     /// <param name="header">
28     /// <para>A header.</para>
29     /// </param>
30     public UInt32LinksTargetsSizeBalancedTreeMethods(LinksConstants<uint> constants,
31     ↪ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
32
33     /// <summary>
34     /// <para>
35     /// Gets the left reference using the specified node.
36     /// </para>
37     /// </summary>
38     /// <param name="node">
39     /// <para>The node.</para>
40     /// </param>
41     /// <returns>
42     /// <para>The ref uint</para>
43     /// </returns>
44     [MethodImpl(MethodImplOptions.AggressiveInlining)]
45     protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsTarget;
46
47     /// <summary>
48     /// <para>
49     /// Gets the right reference using the specified node.
50     /// </para>
51     /// </summary>
52     /// <param name="node">
53     /// <para>The node.</para>
54     /// </param>
55     /// <returns>
56     /// <para>The ref uint</para>
57     /// </returns>
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     protected override ref uint GetRightReference(uint node) => ref
60     ↪ Links[node].RightAsTarget;
61
62     /// <summary>
63     /// <para>
64     /// Gets the left using the specified node.
65     /// </para>
66     /// </summary>
67     /// <param name="node">
68     /// <para>The node.</para>
69     /// </param>
70     /// <returns>
71     /// <para>The uint</para>
72     /// </returns>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected override uint GetLeft(uint node) => Links[node].LeftAsTarget;
75
76     /// <summary>
77     /// <para>
78     /// Gets the right using the specified node.
79     /// </para>
80     /// </summary>
81     /// <param name="node">
82     /// <para>The node.</para>
83     /// </param>
84     /// <returns>
85     /// <para>The uint</para>
86     /// </returns>
87     [MethodImpl(MethodImplOptions.AggressiveInlining)]
88     protected override uint GetRight(uint node) => Links[node].RightAsTarget;
89
90     /// <summary>
91     /// <para>
92     /// Gets the left using the specified node.
93     /// </para>
94     /// </summary>
95     /// <param name="node">
96     /// <para>The node.</para>
97     /// </param>
98     /// <returns>
99     /// <para>The uint</para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override uint GetLeft(uint node) => Links[node].LeftAsTarget;
103
104    /// <summary>
105    /// <para>
106    /// Gets the right using the specified node.
107    /// </para>
108    /// </summary>
109    /// <param name="node">
110    /// <para>The node.</para>
111    /// </param>
112    /// <returns>
113    /// <para>The uint</para>
114    /// </returns>
115    [MethodImpl(MethodImplOptions.AggressiveInlining)]
116    protected override uint GetRight(uint node) => Links[node].RightAsTarget;

```

```

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

```

```

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

```

```

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

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

```

1.96 ./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
    ↪ organizing the storage of links with addresses represented as <see cref="uint" />.</para>
13     /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
    ↪ размером, для организации хранения связей с адресами представленными в виде <see
    ↪ cref="uint"/>.</para>
14     /// </summary>
15     public unsafe class UInt32UnitedMemoryLinks : UnitedMemoryLinksBase<uint>
16     {
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         /// Создает экземпляр базы данных Links в файле по указанному адресу, с указанным
    ↪ минимальным шагом расширения базы данных.

```

```

37     /// </summary>
38     /// <param name="address">Полный путь к файлу базы данных.</param>
39     /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
    ↪ байтах.</param>
40     [MethodImpl(MethodImplOptions.AggressiveInlining)]
41     public UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
    ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
    ↪ memoryReservationStep) { }
42
43     /// <summary>
44     /// <para>
45     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     /// <param name="memory">
50     /// <para>A memory.</para>
51     /// <para></para>
52     /// </param>
53     [MethodImpl(MethodImplOptions.AggressiveInlining)]
54     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
    ↪ DefaultLinksSizeStep) { }
55
56     /// <summary>
57     /// <para>
58     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
59     /// </para>
60     /// <para></para>
61     /// </summary>
62     /// <param name="memory">
63     /// <para>A memory.</para>
64     /// <para></para>
65     /// </param>
66     /// <param name="memoryReservationStep">
67     /// <para>A memory reservation step.</para>
68     /// <para></para>
69     /// </param>
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
    ↪ memoryReservationStep) : this(memory, memoryReservationStep,
    ↪ Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }
72
73     /// <summary>
74     /// <para>
75     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <param name="memory">
80     /// <para>A memory.</para>
81     /// <para></para>
82     /// </param>
83     /// <param name="memoryReservationStep">
84     /// <para>A memory reservation step.</para>
85     /// <para></para>
86     /// </param>
87     /// <param name="constants">
88     /// <para>A constants.</para>
89     /// <para></para>
90     /// </param>
91     /// <param name="indexTreeType">
92     /// <para>A index tree type.</para>
93     /// <para></para>
94     /// </param>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
    ↪ memoryReservationStep, LinksConstants<uint> constants, IndexTreeType indexTreeType)
    ↪ : base(memory, memoryReservationStep, constants)
97     {
98         if (indexTreeType == IndexTreeType.SizeBalancedTree)
99         {
100             _createSourceTreeMethods = () => new
    ↪ UInt32LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
101             _createTargetTreeMethods = () => new
    ↪ UInt32LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
102         }
103         else

```

```

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

```

```

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

```

```

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

```



```

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

```

```

411     /// <para></para>
412     /// </returns>
413     [MethodImpl(MethodImplOptions.AggressiveInlining)]
414     protected override uint Decrement(uint link) => --link;
415 }
416 }

```

1.97 ./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        /// <summary>
27        /// <para>
28        /// The header.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        protected new readonly LinksHeader<ulong>* Header;
33
34        /// <summary>
35        /// <para>
36        /// Initializes a new <see cref="UInt64LinksAvlBalancedTreeMethodsBase"/> instance.
37        /// </para>
38        /// <para></para>
39        /// </summary>
40        /// <param name="constants">
41        /// <para>A constants.</para>
42        /// <para></para>
43        /// </param>
44        /// <param name="links">
45        /// <para>A links.</para>
46        /// <para></para>
47        /// </param>
48        /// <param name="header">
49        /// <para>A header.</para>
50        /// <para></para>
51        /// </param>
52        protected UInt64LinksAvlBalancedTreeMethodsBase(LinksConstants<ulong> constants,
53    ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
54        : base(constants, (byte*)links, (byte*)header)
55        {
56            Links = links;
57            Header = header;
58        }
59
60        /// <summary>
61        /// <para>
62        /// Gets the zero.
63        /// </para>
64        /// <para></para>
65        /// </summary>
66        /// <returns>
67        /// <para>The ulong</para>
68        /// <para></para>
69        /// </returns>
70        [MethodImpl(MethodImplOptions.AggressiveInlining)]
71        protected override ulong GetZero() => 0UL;
72
73        /// <summary>
74        /// <para>
75        /// Determines whether this instance equal to zero.
76        /// </para>
```

```

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

```

```

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

```

```

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

```

```

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

```

```

383 {
384     ref var firstLink = ref Links[first];
385     ref var secondLink = ref Links[second];
386     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
387 }
388
389 /// <summary>
390 /// <para>
391 /// Gets the size value using the specified value.
392 /// </para>
393 /// <para></para>
394 /// </summary>
395 /// <param name="value">
396 /// <para>The value.</para>
397 /// <para></para>
398 /// </param>
399 /// <returns>
400 /// <para>The ulong</para>
401 /// <para></para>
402 /// </returns>
403 [MethodImpl(MethodImplOptions.AggressiveInlining)]
404 protected override ulong GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;
405
406 /// <summary>
407 /// <para>
408 /// Sets the size value using the specified stored value.
409 /// </para>
410 /// <para></para>
411 /// </summary>
412 /// <param name="storedValue">
413 /// <para>The stored value.</para>
414 /// <para></para>
415 /// </param>
416 /// <param name="size">
417 /// <para>The size.</para>
418 /// <para></para>
419 /// </param>
420 [MethodImpl(MethodImplOptions.AggressiveInlining)]
421 protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
    ↪ storedValue & 31UL | (size & 134217727UL) << 5;
422
423 /// <summary>
424 /// <para>
425 /// Determines whether this instance get left is child value.
426 /// </para>
427 /// <para></para>
428 /// </summary>
429 /// <param name="value">
430 /// <para>The value.</para>
431 /// <para></para>
432 /// </param>
433 /// <returns>
434 /// <para>The bool</para>
435 /// <para></para>
436 /// </returns>
437 [MethodImpl(MethodImplOptions.AggressiveInlining)]
438 protected override bool GetLeftIsChildValue(ulong value) => (value & 16UL) >> 4 == 1UL;
439
440 /// <summary>
441 /// <para>
442 /// Sets the left is child value using the specified stored value.
443 /// </para>
444 /// <para></para>
445 /// </summary>
446 /// <param name="storedValue">
447 /// <para>The stored value.</para>
448 /// <para></para>
449 /// </param>
450 /// <param name="value">
451 /// <para>The value.</para>
452 /// <para></para>
453 /// </param>
454 [MethodImpl(MethodImplOptions.AggressiveInlining)]
455 protected override void SetLeftIsChildValue(ref ulong storedValue, bool value) =>
    ↪ storedValue = storedValue & 4294967279UL | (As<bool, byte>(ref value) & 1UL) << 4;
456
457 /// <summary>

```



```

458    /// <para>
459    /// Determines whether this instance get right is child value.
460    /// </para>
461    /// <para></para>
462    /// </summary>
463    /// <param name="value">
464    /// <para>The value.</para>
465    /// <para></para>
466    /// </param>
467    /// <returns>
468    /// <para>The bool</para>
469    /// <para></para>
470    /// </returns>
471    [MethodImpl(MethodImplOptions.AggressiveInlining)]
472    protected override bool GetRightIsChildValue(ulong value) => (value & 8UL) >> 3 == 1UL;
473
474    /// <summary>
475    /// <para>
476    /// Sets the right is child value using the specified stored value.
477    /// </para>
478    /// <para></para>
479    /// </summary>
480    /// <param name="storedValue">
481    /// <para>The stored value.</para>
482    /// <para></para>
483    /// </param>
484    /// <param name="value">
485    /// <para>The value.</para>
486    /// <para></para>
487    /// </param>
488    [MethodImpl(MethodImplOptions.AggressiveInlining)]
489    protected override void SetRightIsChildValue(ref ulong storedValue, bool value) =>
490        ↪ storedValue = storedValue & 4294967287UL | (As<bool, byte>(ref value) & 1UL) << 3;
491
492    /// <summary>
493    /// <para>
494    /// Gets the balance value using the specified value.
495    /// </para>
496    /// <para></para>
497    /// </summary>
498    /// <param name="value">
499    /// <para>The value.</para>
500    /// <para></para>
501    /// </param>
502    /// <returns>
503    /// <para>The sbyte</para>
504    /// <para></para>
505    /// </returns>
506    [MethodImpl(MethodImplOptions.AggressiveInlining)]
507    protected override sbyte GetBalanceValue(ulong value) => unchecked((sbyte)(value & 7UL |
508        ↪ 0xF8UL * ((value & 4UL) >> 2))); // if negative, then continue ones to the end of
509        ↪ sbyte
510
511    /// <summary>
512    /// <para>
513    /// Sets the balance value using the specified stored value.
514    /// </para>
515    /// <para></para>
516    /// </summary>
517    /// <param name="storedValue">
518    /// <para>The stored value.</para>
519    /// <para></para>
520    /// </param>
521    /// <param name="value">
522    /// <para>The value.</para>
523    /// <para></para>
524    /// </param>
525    [MethodImpl(MethodImplOptions.AggressiveInlining)]
526    protected override void SetBalanceValue(ref ulong storedValue, sbyte value) =>
527        ↪ storedValue = unchecked(storedValue & 4294967288UL | (ulong)((byte)value >> 5 & 4 |
528        ↪ value & 3) & 7UL);
529
530    /// <summary>
531    /// <para>
532    /// Gets the header reference.
533    /// </para>
534    /// <para></para>
535    /// </summary>

```

```

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

```

1.99 ./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         /// <summary>
26         /// <para>
27         /// The header.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         protected new readonly LinksHeader<ulong>* Header;
32
33         /// <summary>
34         /// <para>
35         /// Initializes a new <see cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase" />
36         ↪ instance.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="constants">
41         /// <para>A constants.</para>
42         /// <para></para>
43         /// </param>
44         /// <param name="links">
45         /// <para>A links.</para>
46         /// <para></para>
47         /// </param>
48         /// <param name="header">
49         /// <para>A header.</para>
50         /// <para></para>
51         /// </param>

```

```

50     protected UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
    ↪     constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
51         : base(constants, (byte*)links, (byte*)header)
52     {
53         Links = links;
54         Header = header;
55     }
56
57     /// <summary>
58     /// <para>
59     /// Gets the zero.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     /// <returns>
64     /// <para>The ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ulong GetZero() => OUL;
69
70     /// <summary>
71     /// <para>
72     /// Determines whether this instance equal to zero.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="value">
77     /// <para>The value.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The bool</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override bool EqualToZero(ulong value) => value == OUL;
86
87     /// <summary>
88     /// <para>
89     /// Determines whether this instance are equal.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="first">
94     /// <para>The first.</para>
95     /// <para></para>
96     /// </param>
97     /// <param name="second">
98     /// <para>The second.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The bool</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    protected override bool AreEqual(ulong first, ulong second) => first == second;
107
108    /// <summary>
109    /// <para>
110    /// Determines whether this instance greater than zero.
111    /// </para>
112    /// <para></para>
113    /// </summary>
114    /// <param name="value">
115    /// <para>The value.</para>
116    /// <para></para>
117    /// </param>
118    /// <returns>
119    /// <para>The bool</para>
120    /// <para></para>
121    /// </returns>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    protected override bool GreaterThanZero(ulong value) => value > OUL;
124
125    /// <summary>
126    /// <para>

```

```

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

```

```

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

```

```

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

```

1.100 ./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 :
    ↳ LinksSizeBalancedTreeMethodsBase<ulong>
16  {
17      /// <summary>
18      /// <para>
19      /// The links.
20      /// </para>
21      /// <para></para>
22      /// </summary>
23      protected new readonly RawLink<ulong>* Links;
24      /// <summary>
25      /// <para>
26      /// The header.
27      /// </para>
28      /// <para></para>
29      /// </summary>
30      protected new readonly LinksHeader<ulong>* Header;
31
32      /// <summary>
33      /// <para>
34      /// Initializes a new <see cref="UInt64LinksSizeBalancedTreeMethodsBase"/> instance.
35      /// </para>
36      /// <para></para>
37      /// </summary>
38      /// <param name="constants">
39      /// <para>A constants.</para>
40      /// <para></para>
41      /// </param>
42      /// <param name="links">
43      /// <para>A links.</para>
44      /// <para></para>
45      /// </param>
46      /// <param name="header">
47      /// <para>A header.</para>
48      /// <para></para>
49      /// </param>
50      protected UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header)
    : base(constants, (byte*)links, (byte*)header)
51      {
52          Links = links;
53          Header = header;
54      }
55
56
57      /// <summary>
58      /// <para>
59      /// Gets the zero.
60      /// </para>
61      /// <para></para>
62      /// </summary>
63      /// <returns>
64      /// <para>The ulong</para>
65      /// <para></para>
66      /// </returns>
67      [MethodImpl(MethodImplOptions.AggressiveInlining)]
68      protected override ulong GetZero() => OUL;
69
70      /// <summary>
71      /// <para>
72      /// Determines whether this instance equal to zero.
73      /// </para>
74      /// <para></para>
75      /// </summary>
76      /// <param name="value">
77      /// <para>The value.</para>
78      /// <para></para>
79      /// </param>
80      /// <returns>
81      /// <para>The bool</para>
82      /// <para></para>
83      /// </returns>
84      [MethodImpl(MethodImplOptions.AggressiveInlining)]
85      protected override bool EqualToZero(ulong value) => value == OUL;
86
87      /// <summary>
88      /// <para>
89      /// Determines whether this instance are equal.

```



```

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

```

```

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

```

```

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

```

```

321    /// <param name="first">
322    /// <para>The first.</para>
323    /// <para></para>
324    /// </param>
325    /// <param name="second">
326    /// <para>The second.</para>
327    /// <para></para>
328    /// </param>
329    /// <returns>
330    /// <para>The ulong</para>
331    /// <para></para>
332    /// </returns>
333    [MethodImpl(MethodImplOptions.AggressiveInlining)]
334    protected override ulong Subtract(ulong first, ulong second) => first - second;
335
336    /// <summary>
337    /// <para>
338    /// Determines whether this instance first is to the left of second.
339    /// </para>
340    /// <para></para>
341    /// </summary>
342    /// <param name="first">
343    /// <para>The first.</para>
344    /// <para></para>
345    /// </param>
346    /// <param name="second">
347    /// <para>The second.</para>
348    /// <para></para>
349    /// </param>
350    /// <returns>
351    /// <para>The bool</para>
352    /// <para></para>
353    /// </returns>
354    [MethodImpl(MethodImplOptions.AggressiveInlining)]
355    protected override bool FirstIsToTheLeftOfSecond(ulong first, ulong second)
356    {
357        ref var firstLink = ref Links[first];
358        ref var secondLink = ref Links[second];
359        return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
360            ↪ secondLink.Source, secondLink.Target);
361    }
362
363    /// <summary>
364    /// <para>
365    /// Determines whether this instance first is to the right of second.
366    /// </para>
367    /// <para></para>
368    /// </summary>
369    /// <param name="first">
370    /// <para>The first.</para>
371    /// <para></para>
372    /// </param>
373    /// <param name="second">
374    /// <para>The second.</para>
375    /// <para></para>
376    /// </param>
377    /// <returns>
378    /// <para>The bool</para>
379    /// <para></para>
380    /// </returns>
381    [MethodImpl(MethodImplOptions.AggressiveInlining)]
382    protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
383    {
384        ref var firstLink = ref Links[first];
385        ref var secondLink = ref Links[second];
386        return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
387            ↪ secondLink.Source, secondLink.Target);
388    }
389
390    /// <summary>
391    /// <para>
392    /// Gets the header reference.
393    /// </para>
394    /// <para></para>
395    /// </summary>
396    /// <returns>
397    /// <para>A ref links header of ulong</para>
398    /// <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>
35         public UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
36             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37             ↳ { }
38
39         /// <summary>
40         /// <para>
41         /// Gets the left reference using the specified node.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="node">
46         /// <para>The node.</para>
47         /// <para></para>
48         /// </param>
49         /// <returns>
50         /// <para>The ref ulong</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

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

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

185     [MethodImpl(MethodImplOptions.AggressiveInlining)]
186     //protected override bool GetLeftIsChild(ulong node) => IsChild(node, GetLeft(node));

187     /// <summary>
188     /// <para>
189     /// Sets the left is child using the specified node.
190     /// </para>
191     /// <para></para>
192     /// </summary>
193     /// <param name="node">
194     /// <para>The node.</para>
195     /// <para></para>
196     /// </param>

```

```

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

```



```

275     /// <para></para>
276     /// </param>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
    ↪ Links[node].SizeAsSource, value);
279
280     /// <summary>
281     /// <para>
282     /// Gets the tree root.
283     /// </para>
284     /// <para></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,
    ↪ ulong secondSource, ulong secondTarget)
338     => firstSource < secondSource || (firstSource == secondSource && firstTarget <
    ↪ secondTarget);
339
340     /// <summary>
341     /// <para>
342     /// Determines whether this instance first is to the right of second.
343     /// </para>
344     /// <para></para>
345     /// </summary>
346     /// <param name="firstSource">
347     /// <para>The first source.</para>
348     /// <para></para>
349     /// </param>

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 64 links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods :
15    ↪     UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↪     cref="UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="links">
29        /// <para>A links.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="header">
33        /// <para>A header.</para>
34        /// <para></para>
35        /// </param>

```

```

33     /// </param>
34     public UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
    ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
    ↪ links, header) { }
35
36     /// <summary>
37     /// <para>
38     /// Gets the left reference using the specified node.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     /// <param name="node">
43     /// <para>The node.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>The ref ulong</para>
48     /// <para></para>
49     /// </returns>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     protected override ref ulong GetLeftReference(ulong node) => ref
    ↪ Links[node].LeftAsSource;
52
53     /// <summary>
54     /// <para>
55     /// Gets the right reference using the specified node.
56     /// </para>
57     /// <para></para>
58     /// </summary>
59     /// <param name="node">
60     /// <para>The node.</para>
61     /// <para></para>
62     /// </param>
63     /// <returns>
64     /// <para>The ref ulong</para>
65     /// <para></para>
66     /// </returns>
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     protected override ref ulong GetRightReference(ulong node) => ref
    ↪ Links[node].RightAsSource;
69
70     /// <summary>
71     /// <para>
72     /// Gets the left using the specified node.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="node">
77     /// <para>The node.</para>
78     /// <para></para>
79     /// </param>
80     /// <returns>
81     /// <para>The ulong</para>
82     /// <para></para>
83     /// </returns>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
86
87     /// <summary>
88     /// <para>
89     /// Gets the right using the specified node.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// <para></para>
96     /// </param>
97     /// <returns>
98     /// <para>The ulong</para>
99     /// <para></para>
100    /// </returns>
101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
102    protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
103
104    /// <summary>
105    /// <para>
106    /// Sets the left using the specified node.

```

```

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

```

```

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

```

```

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 :
15    ↳ UInt64LinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt64LinksSourcesSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="header">
32        /// <para>A header.</para>
33        /// <para></para>
34        /// </param>
35        public UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
36    ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
37    ↳ { }
38
39    /// <summary>
40    /// <para>
41    /// Gets the left reference using the specified node.
42    /// </para>
43    /// <para></para>
44    /// </summary>
45    /// <param name="node">
46    /// <para>The node.</para>
47    /// <para></para>
48    /// </param>
49    /// <returns>
50    /// <para>The ref ulong</para>

```

```

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

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

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

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

```

```

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

```



```

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

```

```

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

```

1.104 ./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>
48         /// </param>
49         /// <returns>
50         /// <para>The ref ulong</para>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref ulong GetLeftReference(ulong node) => ref
55             ↳ Links[node].LeftAsTarget;
56
57         /// <summary>
58         /// <para>
59         /// Gets the right reference using the specified node.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="node">
64         /// <para>The node.</para>
65         /// <para></para>
66         /// </param>

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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 :
       ↪ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see
       ↪ cref="UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 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>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
262     ↪     ulong secondSource, ulong secondTarget)
263     => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
264     ↪     secondSource);
265
266     /// <summary>
267     /// <para>
268     /// Clears the node using the specified node.
269     /// </para>
270     /// <para></para>
271     /// </summary>
272     /// <param name="node">
273     /// <para>The node.</para>
274     /// <para></para>
275     /// </param>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void ClearNode(ulong node)

```

```

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         /// </param>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>
32         /// <para></para>
33         /// </param>
34         public UInt64LinksTargetsSizeBalancedTreeMethods(LinksConstants<ulong> constants,
35             ↳ RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
36             ↳ { }
37
38         /// <summary>
39         /// <para>
40         /// Gets the left reference using the specified node.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         /// <param name="node">
45         /// <para>The node.</para>
46         /// <para></para>
47         /// </param>
48         /// <returns>
49         /// <para>The ref ulong</para>
50         /// <para></para>
51         /// </returns>
52         [MethodImpl(MethodImplOptions.AggressiveInlining)]
53         protected override ref ulong GetLeftReference(ulong node) => ref
54             ↳ Links[node].LeftAsTarget;
55
56         /// <summary>
57         /// <para>
58         /// Gets the right reference using the specified node.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         /// <param name="node">
63         /// <para>The node.</para>
64         /// <para></para>
65         /// </param>
66         /// <returns>
67         /// <para>The ref ulong</para>
68         /// <para></para>
69         /// </returns>

```

```

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

```

```

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

```

```

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

```

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         /// <para></para>
35         /// </param>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public UInt64UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
38
39         /// <summary>
40         /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
41         ↪ минимальным шагом расширения базы данных.
42         /// </summary>
43         /// <param name="address">Полный путь к файлу базы данных.</param>
44         /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
45         ↪ байтах.</param>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public UInt64UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
48         ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
49         ↪ memoryReservationStep) { }
50
51         /// <summary>
52         /// <para>
53         /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         /// <param name="memory">
58         /// <para>A memory.</para>
59         /// <para></para>
60         /// </param>
61         [MethodImpl(MethodImplOptions.AggressiveInlining)]
62         public UInt64UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
63         ↪ DefaultLinksSizeStep) { }
64
65         /// <summary>
66         /// <para>
67         /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         /// <param name="memory">
72         /// <para>A memory.</para>
73         /// <para></para>
74         /// </param>
75         /// <param name="memoryReservationStep">
76         /// <para>A memory reservation step.</para>
77         /// <para></para>
78         /// </param>
79         [MethodImpl(MethodImplOptions.AggressiveInlining)]
80         public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
81         ↪ memoryReservationStep) : this(memory, memoryReservationStep,
82         ↪ Default<LinksConstants<ulong>>.Instance, IndexTreeType.Default) { }
83
84         /// <summary>
85         /// <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,
18     ↪ TLink>
19     {
20         private static readonly EqualityComparer<TLink> _equalityComparer =
21         ↪ EqualityComparer<TLink>.Default;
22
23         /// <summary>
24         /// <para>
25         /// Initializes a new <see cref="PropertiesOperator" /> instance.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         /// <param name="links">
30         /// <para>A links.</para>
31         /// <para></para>
32         /// </param>
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         public PropertiesOperator(ILinks<TLink> links) : base(links) { }
35     }
36 }

```

```

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      private readonly TLink _propertyMarker;
22      private readonly TLink _propertyValueMarker;
23
24      /// <summary>
25      /// <para>
26      /// Initializes a new <see cref="PropertyOperator" /> instance.
27      /// </para>
28      /// </summary>
29      /// <param name="links">
30      /// <para>A links.</para>
31      /// </param>
32      /// <param name="propertyMarker">
33      /// <para>A property marker.</para>
34      /// </param>
35      /// <param name="propertyValueMarker">
36      /// <para>A property value marker.</para>
37      /// </param>
38      [MethodImpl(MethodImplOptions.AggressiveInlining)]
39      public PropertyOperator(ILinks<TLink> links, TLink propertyMarker, TLink
40          ↳ propertyValueMarker) : base(links)
41      {
42          _propertyMarker = propertyMarker;
43          _propertyValueMarker = propertyValueMarker;
44      }
45
46      /// <summary>
47      /// <para>
48      /// Gets the link.
49      /// </para>
50      /// </summary>
51      /// <param name="link">
52      /// <para>The link.</para>
53      /// </param>
54      /// <returns>
55      /// <para>The link</para>
56      /// </returns>
57      [MethodImpl(MethodImplOptions.AggressiveInlining)]
58      public TLink Get(TLink link)
59      {
60          var property = _links.SearchOrDefault(link, _propertyMarker);
61          return GetValue(GetContainer(property));
62      }
63
64      [MethodImpl(MethodImplOptions.AggressiveInlining)]
65      private TLink GetContainer(TLink property)
66      {
67          var valueContainer = default(TLink);
68          if (_equalityComparer.Equals(property, default))
69          {
70              return valueContainer;
71          }
72          var links = _links;
73          var constants = links.Constants;
74          var continueConstant = constants.Continue;
75          var breakConstant = constants.Break;
76          var anyConstant = constants.Any;
77          var query = new Link<TLink>(anyConstant, property, anyConstant);
78          links.Each(candidate =>
79          {
80              var candidateTarget = links.GetTarget(candidate);
81              var valueTarget = links.GetTarget(candidateTarget);
82              if (_equalityComparer.Equals(valueTarget, _propertyValueMarker))
83              {
84                  valueContainer = links.GetIndex(candidate);
85              }
86          }
87      }
88  }

```

```

89         return breakConstant;
90     }
91     return countinueConstant;
92 }, query);
93 return valueContainer;
94 }
95 [MethodImpl(MethodImplOptions.AggressiveInlining)]
96 private TLink GetValue(TLink container) => _equalityComparer.Equals(container, default)
97     ↪ ? default : _links.GetTarget(container);
98
99 /// <summary>
100 /// <para>
101 /// Sets the link.
102 /// </para>
103 /// <para></para>
104 /// </summary>
105 /// <param name="link">
106 /// <para>The link.</para>
107 /// <para></para>
108 /// </param>
109 /// <param name="value">
110 /// <para>The value.</para>
111 /// <para></para>
112 /// </param>
113 [MethodImpl(MethodImplOptions.AggressiveInlining)]
114 public void Set(TLink link, TLink value)
115 {
116     var links = _links;
117     var property = links.GetOrCreate(link, _propertyMarker);
118     var container = GetContainer(property);
119     if (_equalityComparer.Equals(container, default))
120     {
121         links.GetOrCreate(property, value);
122     }
123     else
124     {
125         links.Update(container, property, value);
126     }
127 }
128 }

```

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         private readonly TLink _stack;
22
23         /// <summary>
24         /// <para>
25         /// Gets the is empty value.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         public bool IsEmpty
30         {
31             [MethodImpl(MethodImplOptions.AggressiveInlining)]
32             get => _equalityComparer.Equals(Peek(), _stack);
33         }
34
35         /// <summary>
36         /// <para>

```



```

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

```

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>

```

```

116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 public TLinkAddress Count(ICollection<TLinkAddress> restriction) =>
    ↳ SyncRoot.ExecuteReadOperation(restriction, Unsync.Count);
118
119 /// <summary>
120 /// <para>
121 /// Eaches the handler.
122 /// </para>
123 /// <para></para>
124 /// </summary>
125 /// <param name="handler">
126 /// <para>The handler.</para>
127 /// <para></para>
128 /// </param>
129 /// <param name="restrictions">
130 /// <para>The restrictions.</para>
131 /// <para></para>
132 /// </param>
133 /// <returns>
134 /// <para>The link address</para>
135 /// <para></para>
136 /// </returns>
137 [MethodImpl(MethodImplOptions.AggressiveInlining)]
138 public TLinkAddress Each(Func<ICollection<TLinkAddress>, TLinkAddress> handler,
    ↳ ICollection<TLinkAddress> restrictions) => SyncRoot.ExecuteReadOperation(handler,
    ↳ restrictions, (handler1, restrictions1) => Unsync.Each(handler1, restrictions1));
139
140 /// <summary>
141 /// <para>
142 /// Creates the restrictions.
143 /// </para>
144 /// <para></para>
145 /// </summary>
146 /// <param name="restrictions">
147 /// <para>The restrictions.</para>
148 /// <para></para>
149 /// </param>
150 /// <returns>
151 /// <para>The link address</para>
152 /// <para></para>
153 /// </returns>
154 [MethodImpl(MethodImplOptions.AggressiveInlining)]
155 public TLinkAddress Create(ICollection<TLinkAddress> restrictions) =>
    ↳ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Create);
156
157 /// <summary>
158 /// <para>
159 /// Updates the restrictions.
160 /// </para>
161 /// <para></para>
162 /// </summary>
163 /// <param name="restrictions">
164 /// <para>The restrictions.</para>
165 /// <para></para>
166 /// </param>
167 /// <param name="substitution">
168 /// <para>The substitution.</para>
169 /// <para></para>
170 /// </param>
171 /// <returns>
172 /// <para>The link address</para>
173 /// <para></para>
174 /// </returns>
175 [MethodImpl(MethodImplOptions.AggressiveInlining)]
176 public TLinkAddress Update(ICollection<TLinkAddress> restrictions, ICollection<TLinkAddress>
    ↳ substitution) => SyncRoot.ExecuteWriteOperation(restrictions, substitution,
    ↳ Unsync.Update);
177
178 /// <summary>
179 /// <para>
180 /// Deletes the restrictions.
181 /// </para>
182 /// <para></para>
183 /// </summary>
184 /// <param name="restrictions">
185 /// <para>The restrictions.</para>
186 /// <para></para>

```

```

187     /// </param>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     public void Delete(IList<TLinkAddress> restrictions) =>
190         ↳ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Delete);
191
192     //public T Trigger(IList<T> restriction, Func<IList<T>, IList<T>, T> matchedHandler,
193     ↳ IList<T> substitution, Func<IList<T>, IList<T>, T> substitutedHandler)
194     //{
195     //    if (restriction != null && substitution != null &&
196     ↳ !substitution.EqualTo(restriction))
197     //        return SyncRoot.ExecuteWriteOperation(restriction, matchedHandler,
198     ↳ substitution, substitutedHandler, Unsync.Trigger);
199     //    return SyncRoot.ExecuteReadOperation(restriction, matchedHandler, substitution,
200     ↳ substitutedHandler, Unsync.Trigger);
201     //}
202 }
203 }

```

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         }
62     }
63 }

```

```

59     }
60     return false;
61 }
62
63 /// <summary>
64 /// <para>
65 /// Formats the structure using the specified links.
66 /// </para>
67 /// <para></para>
68 /// </summary>
69 /// <param name="links">
70 /// <para>The links.</para>
71 /// <para></para>
72 /// </param>
73 /// <param name="linkIndex">
74 /// <para>The link index.</para>
75 /// <para></para>
76 /// </param>
77 /// <param name="isElement">
78 /// <para>The is element.</para>
79 /// <para></para>
80 /// </param>
81 /// <param name="renderIndex">
82 /// <para>The render index.</para>
83 /// <para></para>
84 /// </param>
85 /// <param name="renderDebug">
86 /// <para>The render debug.</para>
87 /// <para></para>
88 /// </param>
89 /// <returns>
90 /// <para>The string</para>
91 /// <para></para>
92 /// </returns>
93 [MethodImpl(MethodImplOptions.AggressiveInlining)]
94 public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
  → Func<Link<ulong>, bool> isElement, bool renderIndex = false, bool renderDebug =
  → false)
95 {
96     var sb = new StringBuilder();
97     var visited = new HashSet<ulong>();
98     links.AppendStructure(sb, visited, linkIndex, isElement, (innerSb, link) =>
  → innerSb.Append(link.Index), renderIndex, renderDebug);
99     return sb.ToString();
100 }
101
102 /// <summary>
103 /// <para>
104 /// Formats the structure using the specified links.
105 /// </para>
106 /// <para></para>
107 /// </summary>
108 /// <param name="links">
109 /// <para>The links.</para>
110 /// <para></para>
111 /// </param>
112 /// <param name="linkIndex">
113 /// <para>The link index.</para>
114 /// <para></para>
115 /// </param>
116 /// <param name="isElement">
117 /// <para>The is element.</para>
118 /// <para></para>
119 /// </param>
120 /// <param name="appendElement">
121 /// <para>The append element.</para>
122 /// <para></para>
123 /// </param>
124 /// <param name="renderIndex">
125 /// <para>The render index.</para>
126 /// <para></para>
127 /// </param>
128 /// <param name="renderDebug">
129 /// <para>The render debug.</para>
130 /// <para></para>
131 /// </param>
132 /// <returns>
133 /// <para>The string</para>

```

```

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

```

```

204         if (renderIndex)
205         {
206             sb.Append(link.Index);
207             sb.Append(':');
208         }
209         if (link.Source == link.Index)
210         {
211             sb.Append(link.Index);
212         }
213         else
214         {
215             var source = new Link<ulong>(links.GetLink(link.Source));
216             if (isElement(source))
217             {
218                 appendElement(sb, source);
219             }
220             else
221             {
222                 links.AppendStructure(sb, visited, source.Index, isElement,
223                     ↪ appendElement, renderIndex);
224             }
225             sb.Append(' ');
226             if (link.Target == link.Index)
227             {
228                 sb.Append(link.Index);
229             }
230             else
231             {
232                 var target = new Link<ulong>(links.GetLink(link.Target));
233                 if (isElement(target))
234                 {
235                     appendElement(sb, target);
236                 }
237                 else
238                 {
239                     links.AppendStructure(sb, visited, target.Index, isElement,
240                         ↪ appendElement, renderIndex);
241                 }
242             }
243             sb.Append(')');
244         }
245         else
246         {
247             if (renderDebug)
248             {
249                 sb.Append('*');
250             }
251             sb.Append(linkIndex);
252         }
253     }
254     else
255     {
256         if (renderDebug)
257         {
258             sb.Append('~');
259         }
260         sb.Append(linkIndex);
261     }
262 }
263 }

```

1.115 ./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         /// Или
49         ///
50         /// public struct TransitionHeader
51         /// {
52         ///     public ulong TransactionIdCombined;
53         ///     public ulong TimestampCombined;
54         ///
55         ///     public ulong TransactionId
56         ///     {
57         ///         get
58         ///         {
59         ///             return (ulong) mask & TransactionIdCombined;
60         ///         }
61         ///     }
62         ///
63         ///     public UniqueTimestamp Timestamp
64         ///     {
65         ///         get
66         ///         {
67         ///             return (UniqueTimestamp)mask & TransactionIdCombined;
68         ///         }
69         ///     }
70         ///
71         ///     public TransactionItemType Type
72         ///     {
73         ///         get
74         ///         {
75         ///             // Использовать по одному биту из TransactionId и Timestamp,
76         ///             // для значения в 2 бита, которое представляет тип операции
77         ///             throw new NotImplementedException();
78         ///         }
79         ///     }
80         /// }
81         ///
82         /// private struct Transition
83         /// {
84         ///     public TransitionHeader Header;
85         ///     public Link Source;
86         ///     public Link Linker;
87         ///     public Link Target;
88         /// }
89         ///
90         /// </remarks>
91         public struct Transition : IEquatable<Transition>
92         {
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
155     ↪ transactionId, Link<ulong> before, Link<ulong> after)
156     {
157         TransactionId = transactionId;
158         Before = before;
159         After = after;
160         Timestamp = uniqueTimestampFactory.Create();
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     /// <summary>
183     /// <para>
184     /// Initializes a new <see cref="Transition"/> instance.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="uniqueTimestampFactory">
189     /// <para>A unique timestamp factory.</para>
190     /// <para></para>
191     /// </param>
192     /// <param name="transactionId">
193     /// <para>A transaction id.</para>
194     /// <para></para>
195     /// </param>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]
197     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
        ↳ }

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

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

229     /// <summary>
230     /// <para>
231     /// Gets the hash code.
232     /// </para>
233     /// <para></para>
234     /// </summary>
235     /// <returns>
236     /// <para>The int</para>
237     /// <para></para>
238     /// </returns>
239     [MethodImpl(MethodImplOptions.AggressiveInlining)]
240

```

```

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
    ↳ Target)) и индексов.
270 /// 2. Хранение трансформаций/операций в отдельном хранилище Links, но дополнительно
    ↳ потребуется решить вопрос
271 /// со ссылками на внешние идентификаторы, или как-то иначе решить вопрос с
    ↳ пересечениями идентификаторов.
272 ///
273 /// Где хранить промежуточный список транзакций?
274 ///
275 /// В оперативной памяти:
276 /// Минусы:
277 /// 1. Может усложнить систему, если она будет функционировать самостоятельно,
278 /// так как нужно отдельно выделять память под список трансформаций.
279 /// 2. Выделенной оперативной памяти может не хватить, в том случае,
280 /// если транзакция использует слишком много трансформаций.
281 /// -> Можно использовать жёсткий диск для слишком длинных транзакций.
282 /// -> Максимальный размер списка трансформаций можно ограничить / задать
    ↳ константой.
283 /// 3. При подтверждении транзакции (Commit) все трансформации записываются разом
    ↳ создавая задержку.
284 ///
285 /// На жёстком диске:
286 /// Минусы:
287 /// 1. Длительный отклик, на запись каждой трансформации.
288 /// 2. Лог транзакций дополнительно наполняется отменёнными транзакциями.
289 /// -> Это может решаться упаковкой/исключением дублирующих операций.
290 /// -> Также это может решаться тем, что короткие транзакции вообще
291 /// не будут записываться в случае отката.
292 /// 3. Перед тем как выполнять отмену операций транзакции нужно дождаться пока все
    ↳ операции (трансформации)
293 /// будут записаны в лог.
294 ///
295 /// </remarks>
296 public class Transaction : DisposableBase
297 {
298     private readonly Queue<Transition> _transitions;
299     private readonly UInt64LinksTransactionsLayer _layer;
300     /// <summary>
301     /// <para>
302     /// Gets or sets the is committed value.
303     /// </para>
304     /// <para></para>
305     /// </summary>
306     public bool IsCommitted { get; private set; }
307     /// <summary>
308     /// <para>

```

```

309     /// Gets or sets the is reverted value.
310     /// </para>
311     /// <para></para>
312     /// </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     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     private void Revert()
363     {
364         EnsureTransactionAllowsWriteOperations(this);
365         var transactionsToRevert = new Transition[_transitions.Count];
366         _transitions.CopyTo(transactionsToRevert, 0);
367         for (var i = transactionsToRevert.Length - 1; i >= 0; i--)
368         {
369             _layer.RevertTransition(transactionsToRevert[i]);
370         }
371         IsReverted = true;
372     }
373
374     /// <summary>
375     /// <para>
376     /// Sets the current transaction using the specified layer.
377     /// </para>
378     /// <para></para>
379     /// </summary>
380     /// <param name="layer">
381     /// <para>The layer.</para>
382     /// <para></para>
383     /// </param>
384     /// <param name="transaction">
385     /// <para>The transaction.</para>
386     /// <para></para>

```

```

387     /// </param>
388     [MethodImpl(MethodImplOptions.AggressiveInlining)]
389     public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
390         ↳ Transaction transaction)
391     {
392         layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
393         layer._currentTransactionTransitions = transaction._transitions;
394         layer._currentTransaction = transaction;
395     }
396     /// <summary>
397     /// <para>
398     /// Ensures the transaction allows write operations using the specified transaction.
399     /// </para>
400     /// <para></para>
401     /// </summary>
402     /// <param name="transaction">
403     /// <para>The transaction.</para>
404     /// <para></para>
405     /// </param>
406     /// <exception cref="InvalidOperationException">
407     /// <para>Transation is committed.</para>
408     /// <para></para>
409     /// </exception>
410     /// <exception cref="InvalidOperationException">
411     /// <para>Transation is reverted.</para>
412     /// <para></para>
413     /// </exception>
414     [MethodImpl(MethodImplOptions.AggressiveInlining)]
415     public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
416     {
417         if (transaction.IsReverted)
418         {
419             throw new InvalidOperationException("Transation is reverted.");
420         }
421         if (transaction.IsCommitted)
422         {
423             throw new InvalidOperationException("Transation is committed.");
424         }
425     }
426     /// <summary>
427     /// <para>
428     /// Disposes the manual.
429     /// </para>
430     /// <para></para>
431     /// </summary>
432     /// <param name="manual">
433     /// <para>The manual.</para>
434     /// <para></para>
435     /// </param>
436     /// <param name="wasDisposed">
437     /// <para>The was disposed.</para>
438     /// <para></para>
439     /// </param>
440     [MethodImpl(MethodImplOptions.AggressiveInlining)]
441     protected override void Dispose(bool manual, bool wasDisposed)
442     {
443         if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
444         {
445             if (!IsCommitted && !IsReverted)
446             {
447                 Revert();
448             }
449             _layer.ResetCurrentTransation();
450         }
451     }
452 }
453 }
454
455     /// <summary>
456     /// <para>
457     /// The from seconds.
458     /// </para>
459     /// <para></para>
460     /// </summary>
461     public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
462     private readonly string _logAddress;
463     private readonly FileStream _log;

```

```

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

```

```

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

```



```

615         CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
        ↪         deletedLink, default));
616     }
617     [MethodImpl(MethodImplOptions.AggressiveInlining)]
618     private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
        ↪         _transitions;
619     [MethodImpl(MethodImplOptions.AggressiveInlining)]
620     private void CommitTransition(Transition transition)
621     {
622         if (_currentTransaction != null)
623         {
624             Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
625         }
626         var transitions = GetCurrentTransitions();
627         transitions.Enqueue(transition);
628     }
629     [MethodImpl(MethodImplOptions.AggressiveInlining)]
630     private void RevertTransition(Transition transition)
631     {
632         if (transition.After.IsNull()) // Revert Deletion with Creation
633         {
634             _links.Create();
635         }
636         else if (transition.Before.IsNull()) // Revert Creation with Deletion
637         {
638             _links.Delete(transition.After.Index);
639         }
640         else // Revert Update
641         {
642             _links.Update(new[] { transition.After.Index, transition.Before.Source,
        ↪             transition.Before.Target });
643         }
644     }
645     [MethodImpl(MethodImplOptions.AggressiveInlining)]
646     private void ResetCurrentTransation()
647     {
648         _currentTransactionId = 0;
649         _currentTransactionTransitions = null;
650         _currentTransaction = null;
651     }
652     [MethodImpl(MethodImplOptions.AggressiveInlining)]
653     private void PushTransitions()
654     {
655         if (_log == null || _transitions == null)
656         {
657             return;
658         }
659         for (var i = 0; i < _transitions.Count; i++)
660         {
661             var transition = _transitions.Dequeue();
662
663             _log.Write(transition);
664             _lastCommittedTransition = transition;
665         }
666     }
667     [MethodImpl(MethodImplOptions.AggressiveInlining)]
668     private void TransitionsPusher()
669     {
670         while (!Disposable.IsDisposed && _transitionsPusher != null)
671         {
672             Thread.Sleep(DefaultPushDelay);
673             PushTransitions();
674         }
675     }
676
677     /// <summary>
678     /// <para>
679     /// Begins the transaction.
680     /// </para>
681     /// <para></para>
682     /// </summary>
683     /// <returns>
684     /// <para>The transaction</para>
685     /// <para></para>
686     /// </returns>
687     [MethodImpl(MethodImplOptions.AggressiveInlining)]
688     public Transaction BeginTransaction() => new Transaction(this);
689     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

690 private void DisposeTransitions()
691 {
692     try
693     {
694         var pusher = _transitionsPusher;
695         if (pusher != null)
696         {
697             _transitionsPusher = null;
698             pusher.Wait();
699         }
700         if (_transitions != null)
701         {
702             PushTransitions();
703         }
704         _log.DisposeIfPossible();
705         FileHelpers.WriteFirst(_logAddress, _lastCommittedTransition);
706     }
707     catch (Exception ex)
708     {
709         ex.Ignore();
710     }
711 }
712
713 #region DisposalBase
714
715 /// <summary>
716 /// <para>
717 /// Disposes the manual.
718 /// </para>
719 /// <para></para>
720 /// </summary>
721 /// <param name="manual">
722 /// <para>The manual.</para>
723 /// <para></para>
724 /// </param>
725 /// <param name="wasDisposed">
726 /// <para>The was disposed.</para>
727 /// <para></para>
728 /// </param>
729 [MethodImpl(MethodImplOptions.AggressiveInlining)]
730 protected override void Dispose(bool manual, bool wasDisposed)
731 {
732     if (!wasDisposed)
733     {
734         DisposeTransitions();
735     }
736     base.Dispose(manual, wasDisposed);
737 }
738
739 #endregion
740 }
741 }

```

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     public unsafe static class GenericLinksTests
11     {
12         [Fact]
13         public static void CRUDTest()
14         {
15             Using<byte>(links => links.TestCRUDOperations());
16             Using<ushort>(links => links.TestCRUDOperations());
17             Using<uint>(links => links.TestCRUDOperations());
18             Using<ulong>(links => links.TestCRUDOperations());
19         }
20
21         [Fact]
22         public static void RawNumbersCRUDTest()
23         {
24             Using<byte>(links => links.TestRawNumbersCRUDOperations());
25             Using<ushort>(links => links.TestRawNumbersCRUDOperations());

```

```

26         Using<uint>(links => links.TestRawNumbersCRUDOperations());
27         Using<ulong>(links => links.TestRawNumbersCRUDOperations());
28     }
29
30     [Fact]
31     public static void MultipleRandomCreationsAndDeletionsTest()
32     {
33         Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
            ↪ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
            ↪ implementation of tree cuts out 5 bits from the address space.
34         Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
            ↪ stMultipleRandomCreationsAndDeletions(100));
35         Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
            ↪ MultipleRandomCreationsAndDeletions(100));
36         Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
            ↪ tMultipleRandomCreationsAndDeletions(100));
37     }
38     private static void Using<TLink>(Action<ILinks<TLink>> action)
39     {
40         using (var scope = new Scope<Types<HeapResizableDirectMemory,
            ↪ UnitedMemoryLinks<TLink>>>())
41         {
42             action(scope.Use<ILinks<TLink>>());
43         }
44     }
45 }
46 }

```

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

```

1  using System;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Memory;
4  using Xunit;
5  using Xunit.Abstractions;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public static class ILinksBasicTests
10     {
11         [Fact]
12         public static void DeleteAllUsages()
13         {
14             var mem = new HeapResizableDirectMemory();
15             var links = new UnitedMemoryLinks<uint>(mem);
16
17             var root = links.CreatePoint();
18
19             var a = links.CreatePoint();
20             var b = links.CreatePoint();
21
22             links.CreateAndUpdate(a, root);
23             links.CreateAndUpdate(b, root);
24
25             Assert.Equal(5U, links.Count());
26
27             links.DeleteAllUsages(root);
28
29             Assert.Equal(2U, links.Count());
30         }
31     }
32 }

```

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

```

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

```

```

16         Assert.True(constants.IsExternalReference(minimum));
17         Assert.True(constants.IsExternalReference(maximum));
18     }
19 }
20 }

```

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

```

1  using System.IO;
2  using Xunit;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.United.Specific;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public static class ResizableDirectMemoryLinksTests
10     {
11         private static readonly LinksConstants<ulong> _constants =
12             ↪ Default<LinksConstants<ulong>>.Instance;
13
14         [Fact]
15         public static void BasicFileMappedMemoryTest()
16         {
17             var tempFilename = Path.GetTempFileName();
18             using (var memoryAdapter = new UInt64UnitedMemoryLinks(tempFilename))
19             {
20                 memoryAdapter.TestBasicMemoryOperations();
21             }
22             File.Delete(tempFilename);
23
24             [Fact]
25             public static void BasicHeapMemoryTest()
26             {
27                 using (var memory = new
28                     ↪ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
29                 using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
30                     ↪ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
31                 {
32                     memoryAdapter.TestBasicMemoryOperations();
33                 }
34             }
35             private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
36             {
37                 var link = memoryAdapter.Create();
38                 memoryAdapter.Delete(link);
39
40                 [Fact]
41                 public static void NonexistentReferencesHeapMemoryTest()
42                 {
43                     using (var memory = new
44                         ↪ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
45                     using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
46                         ↪ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
47                     {
48                         memoryAdapter.TestNonexistentReferences();
49                     }
50                 }
51                 private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
52                 {
53                     var link = memoryAdapter.Create();
54                     memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);
55                     var resultLink = _constants.Null;
56                     memoryAdapter.Each(foundLink =>
57                     {
58                         resultLink = foundLink[_constants.IndexPart];
59                         return _constants.Break;
60                     }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
61                     Assert.True(resultLink == link);
62                     Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
63                     memoryAdapter.Delete(link);
64                 }
65             }
66         }
67     }
68 }

```

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

```

1  using Xunit;
2  using Platform.Scopes;

```

```

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

```

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

```

1 using System;
2 using Xunit;
3 using Platform.Memory;
4 using Platform.Data.Doublets.Memory.Split.Generic;
5 using Platform.Data.Doublets.Memory;
6
7 namespace Platform.Data.Doublets.Tests
8 {
9     public unsafe static class SplitMemoryGenericLinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using<byte>(links => links.TestCRUDOperations());
15             Using<ushort>(links => links.TestCRUDOperations());
16             Using<uint>(links => links.TestCRUDOperations());
17             Using<ulong>(links => links.TestCRUDOperations());
18         }
19
20         [Fact]
21         public static void RawNumbersCRUDTest()
22         {

```

```

23     UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
24     UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
25     UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
26     UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
27 }
28
29 [Fact]
30 public static void MultipleRandomCreationsAndDeletionsTest()
31 {
32     Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
        ↳ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
        ↳ implementation of tree cuts out 5 bits from the address space.
33     Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
        ↳ stMultipleRandomCreationsAndDeletions(100));
34     Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
        ↳ MultipleRandomCreationsAndDeletions(100));
35     Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
        ↳ tMultipleRandomCreationsAndDeletions(100));
36 }
37 private static void Using<TLink>(Action<ILinks<TLink>> action)
38 {
39     using (var dataMemory = new HeapResizableDirectMemory())
40     using (var indexMemory = new HeapResizableDirectMemory())
41     using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory))
42     {
43         action(memory);
44     }
45 }
46 private static void UsingWithExternalReferences<TLink>(Action<ILinks<TLink>> action)
47 {
48     var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
49     using (var dataMemory = new HeapResizableDirectMemory())
50     using (var indexMemory = new HeapResizableDirectMemory())
51     using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory,
        ↳ SplitMemoryLinks<TLink>.DefaultLinksSizeStep, constants))
52     {
53         action(memory);
54     }
55 }
56 }
57 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLink = System.UInt32;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryUInt32LinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using(links => links.TestCRUDOperations());
15         }
16
17         [Fact]
18         public static void RawNumbersCRUDTest()
19         {
20             UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
21         }
22
23         [Fact]
24         public static void MultipleRandomCreationsAndDeletionsTest()
25         {
26             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
        ↳ leRandomCreationsAndDeletions(500));
27         }
28         private static void Using(Action<ILinks<TLink>> action)
29         {
30             using (var dataMemory = new HeapResizableDirectMemory())
31             using (var indexMemory = new HeapResizableDirectMemory())
32             using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory))
33             {
34                 action(memory);

```

```

35     }
36 }
37 private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
38 {
39     var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
40     using (var dataMemory = new HeapResizableDirectMemory())
41     using (var indexMemory = new HeapResizableDirectMemory())
42     using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory,
43         ↪ UInt32SplitMemoryLinks.DefaultLinksSizeStep, constants))
44     {
45         action(memory);
46     }
47 }
48 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLink = System.UInt64;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      public unsafe static class SplitMemoryUInt64LinksTests
10     {
11         [Fact]
12         public static void CRUDTest()
13         {
14             Using(links => links.TestCRUDOperations());
15         }
16
17         [Fact]
18         public static void RawNumbersCRUDTest()
19         {
20             UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
21         }
22
23         [Fact]
24         public static void MultipleRandomCreationsAndDeletionsTest()
25         {
26             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(500));
27         }
28         private static void Using(Action<ILinks<TLink>> action)
29         {
30             using (var dataMemory = new HeapResizableDirectMemory())
31             using (var indexMemory = new HeapResizableDirectMemory())
32             using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory))
33             {
34                 action(memory);
35             }
36         }
37         private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
38         {
39             var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
40             using (var dataMemory = new HeapResizableDirectMemory())
41             using (var indexMemory = new HeapResizableDirectMemory())
42             using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
43                 ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, constants))
44             {
45                 action(memory);
46             }
47         }
48     }

```

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

```

1  using System.Collections.Generic;
2  using Xunit;
3  using Platform.Ranges;
4  using Platform.Numbers;
5  using Platform.Random;
6  using Platform.Setters;
7  using Platform.Converters;
8
9  namespace Platform.Data.Doublets.Tests
10 {

```

```

11 public static class TestExtensions
12 {
13     public static void TestCRUDOperations<T>(this ILinks<T> links)
14     {
15         var constants = links.Constants;
16
17         var equalityComparer = EqualityComparer<T>.Default;
18
19         var zero = default(T);
20         var one = Arithmetic.Increment(zero);
21
22         // Create Link
23         Assert.True(equalityComparer.Equals(links.Count(), zero));
24
25         var setter = new Setter<T>(constants.Null);
26         links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
27
28         Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
29
30         var linkAddress = links.Create();
31
32         var link = new Link<T>(links.GetLink(linkAddress));
33
34         Assert.True(link.Count == 3);
35         Assert.True(equalityComparer.Equals(link.Index, linkAddress));
36         Assert.True(equalityComparer.Equals(link.Source, constants.Null));
37         Assert.True(equalityComparer.Equals(link.Target, constants.Null));
38
39         Assert.True(equalityComparer.Equals(links.Count(), one));
40
41         // Get first link
42         setter = new Setter<T>(constants.Null);
43         links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
44
45         Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
46
47         // Update link to reference itself
48         links.Update(linkAddress, linkAddress, linkAddress);
49
50         link = new Link<T>(links.GetLink(linkAddress));
51
52         Assert.True(equalityComparer.Equals(link.Source, linkAddress));
53         Assert.True(equalityComparer.Equals(link.Target, linkAddress));
54
55         // Update link to reference null (prepare for delete)
56         var updated = links.Update(linkAddress, constants.Null, constants.Null);
57
58         Assert.True(equalityComparer.Equals(updated, linkAddress));
59
60         link = new Link<T>(links.GetLink(linkAddress));
61
62         Assert.True(equalityComparer.Equals(link.Source, constants.Null));
63         Assert.True(equalityComparer.Equals(link.Target, constants.Null));
64
65         // Delete link
66         links.Delete(linkAddress);
67
68         Assert.True(equalityComparer.Equals(links.Count(), zero));
69
70         setter = new Setter<T>(constants.Null);
71         links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
72
73         Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
74     }
75
76     public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
77     {
78         // Constants
79         var constants = links.Constants;
80         var equalityComparer = EqualityComparer<T>.Default;
81
82         var zero = default(T);
83         var one = Arithmetic.Increment(zero);
84         var two = Arithmetic.Increment(one);
85
86         var h106E = new Hybrid<T>(106L, isExternal: true);
87         var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
88         var h108E = new Hybrid<T>(-108L);
89
90         Assert.Equal(106L, h106E.AbsoluteValue);

```



```

91     Assert.Equal(107L, h107E.AbsoluteValue);
92     Assert.Equal(108L, h108E.AbsoluteValue);
93
94     // Create Link (External -> External)
95     var linkAddress1 = links.Create();
96
97     links.Update(linkAddress1, h106E, h108E);
98
99     var link1 = new Link<T>(links.GetLink(linkAddress1));
100
101     Assert.True(equalityComparer.Equals(link1.Source, h106E));
102     Assert.True(equalityComparer.Equals(link1.Target, h108E));
103
104     // Create Link (Internal -> External)
105     var linkAddress2 = links.Create();
106
107     links.Update(linkAddress2, linkAddress1, h108E);
108
109     var link2 = new Link<T>(links.GetLink(linkAddress2));
110
111     Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
112     Assert.True(equalityComparer.Equals(link2.Target, h108E));
113
114     // Create Link (Internal -> Internal)
115     var linkAddress3 = links.Create();
116
117     links.Update(linkAddress3, linkAddress1, linkAddress2);
118
119     var link3 = new Link<T>(links.GetLink(linkAddress3));
120
121     Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
122     Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
123
124     // Search for created link
125     var setter1 = new Setter<T>(constants.Null);
126     links.Each(h106E, h108E, setter1.SetAndReturnFalse);
127
128     Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
129
130     // Search for nonexistent link
131     var setter2 = new Setter<T>(constants.Null);
132     links.Each(h106E, h107E, setter2.SetAndReturnFalse);
133
134     Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
135
136     // Update link to reference null (prepare for delete)
137     var updated = links.Update(linkAddress3, constants.Null, constants.Null);
138
139     Assert.True(equalityComparer.Equals(updated, linkAddress3));
140
141     link3 = new Link<T>(links.GetLink(linkAddress3));
142
143     Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
144     Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
145
146     // Delete link
147     links.Delete(linkAddress3);
148
149     Assert.True(equalityComparer.Equals(links.Count(), two));
150
151     var setter3 = new Setter<T>(constants.Null);
152     links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
153
154     Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
155 }
156
157 public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
↪ links, int maximumOperationsPerCycle)
158 {
159     var comparer = Comparer<TLink>.Default;
160     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
161     var uint64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
162     for (var N = 1; N < maximumOperationsPerCycle; N++)
163     {
164         var random = new System.Random(N);
165         var created = OUL;
166         var deleted = OUL;
167         for (var i = 0; i < N; i++)
168         {
169             var linksCount = addressToUInt64Converter.Convert(links.Count());

```

```

170         var createPoint = random.NextBoolean();
171         if (linksCount >= 2 && createPoint)
172         {
173             var linksAddressRange = new Range<ulong>(1, linksCount);
174             TLink source = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
                ↪ ddressRange));
175             TLink target = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
                ↪ ddressRange));
                ↪ //-V3086
176             var resultLink = links.GetOrCreate(source, target);
177             if (comparer.Compare(resultLink,
                ↪ uInt64ToAddressConverter.Convert(linksCount)) > 0)
178             {
179                 created++;
180             }
181         }
182         else
183         {
184             links.Create();
185             created++;
186         }
187     }
188     Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
189     for (var i = 0; i < N; i++)
190     {
191         TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
192         if (links.Exists(link))
193         {
194             links.Delete(link);
195             deleted++;
196         }
197     }
198     Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
199 }
200 }
201 }
202 }

```

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

```

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

```

```

34         Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
35     }
36 }
37 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLink = System.UInt32;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     public unsafe static class UnitedMemoryUInt32LinksTests
12     {
13         [Fact]
14         public static void CRUDTest()
15         {
16             Using(links => links.TestCRUDOperations());
17         }
18
19         [Fact]
20         public static void RawNumbersCRUDTest()
21         {
22             Using(links => links.TestRawNumbersCRUDOperations());
23         }
24
25         [Fact]
26         public static void MultipleRandomCreationsAndDeletionsTest()
27         {
28             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
29         }
30         private static void Using(Action<ILinks<TLink>> action)
31         {
32             using (var scope = new Scope<Types<HeapResizableDirectMemory,
33                 ↳ UInt32UnitedMemoryLinks>>())
34             {
35                 action(scope.Use<ILinks<TLink>>());
36             }
37         }
38     }
39 }

```

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

```

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

```

```
31     {
32         using (var scope = new Scope<Types<HeapResizableDirectMemory,
33             ↳ UInt64UnitedMemoryLinks>>())
34         {
35             action(scope.Use<ILinks<TLink>>());
36         }
37     }
38 }
```

Index

`./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.cs`, 442
`./csharp/Platform.Data.Doublets.Tests/ILinksBasicTests.cs`, 443
`./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs`, 443
`./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs`, 444
`./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs`, 444
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs`, 445
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs`, 446
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs`, 447
`./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs`, 447
`./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs`, 450
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs`, 451
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt64LinksTests.cs`, 451
`./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`, 53
`./csharp/Platform.Data.Doublets/LinksOperatorBase.cs`, 54
`./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs`, 55
`./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs`, 55
`./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs`, 57
`./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs`, 57
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 59
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs`, 66
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 73
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 77
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 81
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 84
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 88
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs`, 94
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesLinkedListMethods.cs`, 99
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 104
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs`, 108
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 112
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs`, 115
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs`, 119
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs`, 123
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs`, 140
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs`, 143
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs`, 145
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 147
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase.cs`, 153
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 159
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 162
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 166
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 170

./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 174
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 180
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 185
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 187
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 190
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 194
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 197
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 201
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs, 208
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 208
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase.cs, 214
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 220
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods.cs, 224
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 228
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethods.cs, 232
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 236
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.cs, 242
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 247
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 248
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 252
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 255
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 259
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 263
./csharp/Platform.Data.Doublents/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 269
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 270
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 279
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 286
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 292
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 297
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 301
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 305
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 310
./csharp/Platform.Data.Doublents/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 314
./csharp/Platform.Data.Doublents/Memory/United/Generic/UnitedMemoryLinks.cs, 318
./csharp/Platform.Data.Doublents/Memory/United/Generic/UnitedMemoryLinksBase.cs, 320
./csharp/Platform.Data.Doublents/Memory/United/Generic/UnusedLinksListMethods.cs, 333
./csharp/Platform.Data.Doublents/Memory/United/RawLink.cs, 336
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 338
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 344
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 349
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs, 353
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 356
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 360
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 364
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 370
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 371
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 378
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 383
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 389
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 394
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 398
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 402
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 407
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 411
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 414
./csharp/Platform.Data.Doublents/Memory/United/Specific/UInt64UnusedLinksListMethods.cs, 420
./csharp/Platform.Data.Doublents/PropertyOperators/PropertiesOperator.cs, 421
./csharp/Platform.Data.Doublents/PropertyOperators/PropertyOperator.cs, 422
./csharp/Platform.Data.Doublents/Stacks/Stack.cs, 424
./csharp/Platform.Data.Doublents/Stacks/StackExtensions.cs, 425
./csharp/Platform.Data.Doublents/SynchronizedLinks.cs, 426
./csharp/Platform.Data.Doublents/UInt64LinksExtensions.cs, 429
./csharp/Platform.Data.Doublents/UInt64LinksTransactionsLayer.cs, 432