

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        /// <summary>
20        /// <para>
21        /// The default.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        private static readonly EqualityComparer<TLink> _equalityComparer =
26            ↪ EqualityComparer<TLink>.Default;
27
28        /// <summary>
29        /// <para>
30        /// The target to match.
31        /// </para>
32        /// <para></para>
33        /// </summary>
34        private readonly TLink _targetToMatch;
35
36        /// <summary>
37        /// <para>
38        /// Initializes a new <see cref="TargetMatcher" /> instance.
39        /// </para>
40        /// <para></para>
41        /// </summary>
42        /// <param name="links">
43        /// <para>A links.</para>
44        /// <para></para>
45        /// </param>
46        /// <param name="targetToMatch">
47        /// <para>A target to match.</para>
48        /// <para></para>
49        /// </param>
50        [MethodImpl(MethodImplOptions.AggressiveInlining)]
51        public TargetMatcher(ILinks<TLink> links, TLink targetToMatch) : base(links) =>
52            ↪ _targetToMatch = targetToMatch;
53
54        /// <summary>
55        /// <para>
56        /// Determines whether this instance is matched.
57        /// </para>
58        /// <para></para>
59        /// </summary>
60        /// <param name="link">
61        /// <para>The link.</para>
62        /// <para></para>
63        /// </param>
64        /// <returns>
65        /// <para>The bool</para>
66        /// <para></para>
67        /// </returns>
68        [MethodImpl(MethodImplOptions.AggressiveInlining)]
69        public bool IsMatched(TLink link) => _equalityComparer.Equals(_links.GetTarget(link),
70            ↪ _targetToMatch);
71    }
72 }
```

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

```

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

```

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

```

```

23     /// </param>
24     [MethodImpl(MethodImplOptions.AggressiveInlining)]
25     public LinksCascadeUsagesResolver(ILinks<TLink> links) : base(links) { }
26
27     /// <summary>
28     /// <para>
29     /// Deletes the restrictions.
30     /// </para>
31     /// <para></para>
32     /// </summary>
33     /// <param name="restrictions">
34     /// <para>The restrictions.</para>
35     /// <para></para>
36     /// </param>
37     [MethodImpl(MethodImplOptions.AggressiveInlining)]
38     public override void Delete(ICollection<TLink> restrictions)
39     {
40         var linkIndex = restrictions[_constants.IndexPart];
41         // Use Facade (the last decorator) to ensure recursion working correctly
42         _facade.DeleteAllUsages(linkIndex);
43         _links.Delete(linkIndex);
44     }
45 }
46 }

```

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

```

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

```

```

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

```

```

131     /// <para></para>
132     /// </param>
133     /// <returns>
134     /// <para>The link</para>
135     /// <para></para>
136     /// </returns>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public virtual TLink Create(ICollection<TLink> restrictions) => _links.Create(restrictions);
139
140     /// <summary>
141     /// <para>
142     /// Updates the restrictions.
143     /// </para>
144     /// <para></para>
145     /// </summary>
146     /// <param name="restrictions">
147     /// <para>The restrictions.</para>
148     /// <para></para>
149     /// </param>
150     /// <param name="substitution">
151     /// <para>The substitution.</para>
152     /// <para></para>
153     /// </param>
154     /// <returns>
155     /// <para>The link</para>
156     /// <para></para>
157     /// </returns>
158     [MethodImpl(MethodImplOptions.AggressiveInlining)]
159     public virtual TLink Update(ICollection<TLink> restrictions, ICollection<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(ICollection<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>

```

```

31     /// Initializes a new <see cref="DisposableWithMultipleCallsAllowed"/> instance.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     /// <param name="disposal">
36     /// <para>A disposal.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     public DisposableWithMultipleCallsAllowed(Disposal disposal) : base(disposal) { }
41
42     /// <summary>
43     /// <para>
44     /// Gets the allow multiple dispose calls value.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     protected override bool AllowMultipleDisposeCalls
49     {
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         get => true;
52     }
53 }
54
55     /// <summary>
56     /// <para>
57     /// The disposable.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     protected readonly DisposableWithMultipleCallsAllowed Disposable;
62
63     /// <summary>
64     /// <para>
65     /// Initializes a new <see cref="LinksDisposableDecoratorBase"/> instance.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     /// <param name="links">
70     /// <para>A links.</para>
71     /// <para></para>
72     /// </param>
73     [MethodImpl(MethodImplOptions.AggressiveInlining)]
74     protected LinksDisposableDecoratorBase(ILinks<TLink> links) : base(links) => Disposable
75     ↪ = new DisposableWithMultipleCallsAllowed(Dispose);
76
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     ~LinksDisposableDecoratorBase() => Disposable.Destruct();
79
80     /// <summary>
81     /// <para>
82     /// Disposes this instance.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     public void Dispose() => Disposable.Dispose();
88
89     /// <summary>
90     /// <para>
91     /// Disposes the manual.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="manual">
96     /// <para>The manual.</para>
97     /// <para></para>
98     /// </param>
99     /// <param name="wasDisposed">
100    /// <para>The was disposed.</para>
101    /// <para></para>
102    /// </param>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    protected virtual void Dispose(bool manual, bool wasDisposed)
105    {
106        if (!wasDisposed)
107        {
108            _links.DisposeIfPossible();

```

```

108     }
109 }
110 }
111 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
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     /// <para></para>
16     /// </summary>
17     /// <seealso cref="LinksDecoratorBase{TLink}" />
18     public class LinksInnerReferenceExistenceValidator<TLink> : LinksDecoratorBase<TLink>
19     {
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="LinksInnerReferenceExistenceValidator" /> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         public LinksInnerReferenceExistenceValidator(ILinks<TLink> links) : base(links) { }
32
33         /// <summary>
34         /// <para>
35         /// Eaches the handler.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="handler">
40         /// <para>The handler.</para>
41         /// <para></para>
42         /// </param>
43         /// <param name="restrictions">
44         /// <para>The restrictions.</para>
45         /// <para></para>
46         /// </param>
47         /// <returns>
48         /// <para>The link</para>
49         /// <para></para>
50         /// </returns>
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         public override TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
53         {
54             var links = _links;
55             links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
56             return links.Each(handler, restrictions);
57         }
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>

```

```

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(IList<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         /// <summary>
19         /// <para>
20         /// The default.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private static readonly EqualityComparer<TLink> _equalityComparer =
25             ↪ EqualityComparer<TLink>.Default;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="LinksItselfConstantToSelfReferenceResolver"/> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="links">
34         /// <para>A links.</para>
35         /// <para></para>
36         /// </param>
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public LinksItselfConstantToSelfReferenceResolver(ILinks<TLink> links) : base(links) { }
39
40         /// <summary>
41         /// <para>
42         /// Eaches the handler.
43         /// </para>

```



```

43     /// <para></para>
44     /// </summary>
45     /// <param name="handler">
46     /// <para>The handler.</para>
47     /// <para></para>
48     /// </param>
49     /// <param name="restrictions">
50     /// <para>The restrictions.</para>
51     /// <para></para>
52     /// </param>
53     /// <returns>
54     /// <para>The link</para>
55     /// <para></para>
56     /// </returns>
57     [MethodImpl(MethodImplOptions.AggressiveInlining)]
58     public override TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
59     {
60         var constants = _constants;
61         var itselfConstant = constants.Itself;
62         if (!_equalityComparer.Equals(constants.Any, itselfConstant) &&
63             ↪ restrictions.Contains(itselfConstant))
64         {
65             // Itself constant is not supported for Each method right now, skipping execution
66             return constants.Continue;
67         }
68         return _links.Each(handler, restrictions);
69     }
70     /// <summary>
71     /// <para>
72     /// Updates the restrictions.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="restrictions">
77     /// <para>The restrictions.</para>
78     /// <para></para>
79     /// </param>
80     /// <param name="substitution">
81     /// <para>The substitution.</para>
82     /// <para></para>
83     /// </param>
84     /// <returns>
85     /// <para>The link</para>
86     /// <para></para>
87     /// </returns>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
90         ↪ _links.Update(restrictions, _links.ResolveConstantAsSelfReference(_constants.Itself,
91         ↪ restrictions, substitution));
92     }
93 }

```

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>

```

```

23     /// <para></para>
24     /// </param>
25     [MethodImpl(MethodImplOptions.AggressiveInlining)]
26     public LinksNonExistentDependenciesCreator(ILinks<TLink> links) : base(links) { }
27
28     /// <summary>
29     /// <para>
30     /// Updates the restrictions.
31     /// </para>
32     /// <para></para>
33     /// </summary>
34     /// <param name="restrictions">
35     /// <para>The restrictions.</para>
36     /// <para></para>
37     /// </param>
38     /// <param name="substitution">
39     /// <para>The substitution.</para>
40     /// <para></para>
41     /// </param>
42     /// <returns>
43     /// <para>The link</para>
44     /// <para></para>
45     /// </returns>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public override TLink 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(ICollection<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(ICollection<TLink> restrictions, ICollection<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         /// <summary>
18         /// <para>
19         /// The default.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
24
25         /// <summary>
26         /// <para>
27         /// Initializes a new <see cref="LinksUniquenessResolver" /> instance.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         /// <param name="links">
32         /// <para>A links.</para>
33         /// <para></para>
34         /// </param>
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public LinksUniquenessResolver(ICollection<TLink> links) : base(links) { }
37
38         /// <summary>
39         /// <para>
40         /// Updates the restrictions.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         /// <param name="restrictions">
45         /// <para>The restrictions.</para>
46         /// <para></para>
47         /// </param>

```

```

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

```

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(IList<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(IList<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(IList<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(ILinks<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(IList<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] !=
            ↪ newTarget)
88         {
89             links.Update(updatedLink, newSource == itselfConstant ? updatedLink :
            ↪ newSource,
90                             newTarget == itselfConstant ? updatedLink :
            ↪ newTarget);
91         }
92         return updatedLink;
93     }
94     else
95     {
96         return _facade.MergeAndDelete(updatedLink, existedLink);
97     }
98 }
99
100 /// <summary>
101 /// <para>
102 /// Deletes the restrictions.
103 /// </para>
104 /// <para></para>
105 /// </summary>
106 /// <param name="restrictions">
107 /// <para>The restrictions.</para>
108 /// <para></para>
109 /// </param>
110 [MethodImpl(MethodImplOptions.AggressiveInlining)]
111 public override void Delete(ICollection<TLink> restrictions)
112 {
113     var linkIndex = restrictions[_constants.IndexPart];
114     var links = _links;
115     links.EnforceResetValues(linkIndex);
116     _facade.DeleteAllUsages(linkIndex);
117     links.Delete(linkIndex);
118 }
119 }
120 }

```

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

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Decorators
7  {
8      /// <summary>
9      /// <para>Represents a combined decorator that implements the basic logic for interacting
10      ↪ with the links storage for links with addresses represented as <see cref="System.UInt64"
11      ↪ >.</para>
12      /// <para>Представляет комбинированный декоратор, реализующий основную логику по
13      ↪ взаимодействию с хранилищем связей, для связей с адресами представленными в виде <see
14      ↪ cref="System.UInt64"/>.</para>
15      /// </summary>
16      /// <remarks>
17      /// Возможные оптимизации:
18      /// Объединение в одном поле Source и Target с уменьшением до 32 бит.
19      /// + меньше объём БД
20      /// - меньше производительность
21      /// - больше ограничение на количество связей в БД)
22      /// Ленивое хранение размеров поддеревьев (расчитываемое по мере использования БД)
23      /// + меньше объём БД
24      /// - больше сложность

```



```

21 ///
22 /// Текущее теоретическое ограничение на индекс связи, из-за использования 5 бит в размере
    ↳ поддеревьев для AVL баланса и флагов нитей: 2 в степени (64 минус 5 равно 59 ) равно 576
    ↳ 460 752 303 423 488
23 /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами
    ↳ (битовыми строками) - вариант матрицы (выстраиваемой лениво).
24 ///
25 /// Решить отключать ли проверки при компиляции под Release. Т.е. исключения будут
    ↳ выбрасываться только при #if DEBUG
26 /// </remarks>
27 public class UInt64Links : LinksDisposableDecoratorBase<ulong>
28 {
29     /// <summary>
30     /// <para>
31     /// Initializes a new <see cref="UInt64Links"/> instance.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     /// <param name="links">
36     /// <para>A links.</para>
37     /// <para></para>
38     /// </param>
39     [MethodImpl(MethodImplOptions.AggressiveInlining)]
40     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(IList<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(IList<ulong> restrictions, IList<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     }
96 }

```

```

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>
119     /// </summary>
120     /// <param name="restrictions">
121     /// <para>The restrictions.</para>
122     /// <para></para>
123     /// </param>
124     [MethodImpl(MethodImplOptions.AggressiveInlining)]
125     public override void Delete(IList<ulong> restrictions)
126     {
127         var linkIndex = restrictions[_constants.IndexPart];
128         var links = _links;
129         links.EnforceResetValues(linkIndex);
130         _facade.DeleteAllUsages(linkIndex);
131         links.Delete(linkIndex);
132     }
133 }

```

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     /// TODO0: 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         /// <summary>
24         /// <para>
25         /// The default.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         private static readonly EqualityComparer<TLink> _equalityComparer =
30             ↪ EqualityComparer<TLink>.Default;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="UniLinks"/> instance.
35         /// </para>
36         /// <para></para>
37         /// </summary>

```

```

34    /// <param name="links">
35    /// <para>A links.</para>
36    /// <para></para>
37    /// </param>
38    public UniLinks(ILinks<TLink> links) : base(links) { }
39
40    /// <summary>
41    /// <para>
42    /// The transition.
43    /// </para>
44    /// <para></para>
45    /// </summary>
46    private struct Transition
47    {
48        /// <summary>
49        /// <para>
50        /// The before.
51        /// </para>
52        /// <para></para>
53        /// </summary>
54        public IList<TLink> Before;
55        /// <summary>
56        /// <para>
57        /// The after.
58        /// </para>
59        /// <para></para>
60        /// </summary>
61        public IList<TLink> After;
62
63        /// <summary>
64        /// <para>
65        /// Initializes a new <see cref="Transition"/> instance.
66        /// </para>
67        /// <para></para>
68        /// </summary>
69        /// <param name="before">
70        /// <para>A before.</para>
71        /// <para></para>
72        /// </param>
73        /// <param name="after">
74        /// <para>A after.</para>
75        /// <para></para>
76        /// </param>
77        public Transition(IList<TLink> before, IList<TLink> after)
78        {
79            Before = before;
80            After = after;
81        }
82    }
83
84    //public static readonly TLink NullConstant = Use<LinksConstants<TLink>>.Single.Null;
85    //public static readonly IReadOnlyList<TLink> NullLink = new
86    ↪    ReadOnlyCollection<TLink>(new List<TLink> { NullConstant, NullConstant, NullConstant
87    ↪    });
88
89    // TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
90    ↪    (Links-Expression)
91    /// <summary>
92    /// <para>
93    /// Triggers the restriction.
94    /// </para>
95    /// <para></para>
96    /// </summary>
97    /// <param name="restriction">
98    /// <para>The restriction.</para>
99    /// <para></para>
100    /// </param>
101    /// <param name="matchedHandler">
102    /// <para>The matched handler.</para>
103    /// <para></para>
104    /// </param>
105    /// <param name="substitution">
106    /// <para>The substitution.</para>
107    /// <para></para>
108    /// </param>
109    /// <param name="substitutedHandler">
110    /// <para>The substituted handler.</para>
111    /// <para></para>
112    /// </param>

```

```

109 /// </param>
110 /// <returns>
111 /// <para>The link</para>
112 /// <para></para>
113 /// </returns>
114 public TLink Trigger(IList<TLink> restriction, Func<IList<TLink>, IList<TLink>, TLink>
    ↳ matchedHandler, IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
    ↳ substitutedHandler)
115 {
116     ///List<Transition> transitions = null;
117     ///if (!restriction.IsNullOrEmpty())
118     ///{
119     ///    // Есть причина делать проход (чтение)
120     ///    if (matchedHandler != null)
121     ///    {
122     ///        if (!substitution.IsNullOrEmpty())
123     ///        {
124     ///            // restriction => { 0, 0, 0 } | { 0 } // Create
125     ///            // substitution => { itself, 0, 0 } | { itself, itself, itself } //
126     ↳ Create / Update
127     ///            // substitution => { 0, 0, 0 } | { 0 } // Delete
128     ///            transitions = new List<Transition>();
129     ///            if (Equals(substitution[Constants.IndexPart], Constants.Null))
130     ///            {
131     ///                // If index is Null, that means we always ignore every other
132     ↳ value (they are also Null by definition)
133     ///                var matchDecision = matchedHandler(, NullLink);
134     ///                if (Equals(matchDecision, Constants.Break))
135     ///                {
136     ///                    return false;
137     ///                }
138     ///                if (!Equals(matchDecision, Constants.Skip))
139     ///                {
140     ///                    transitions.Add(new Transition(matchedLink, newValue));
141     ///                }
142     ///            }
143     ///            else
144     ///            {
145     ///                Func<T, bool> handler;
146     ///                handler = link =>
147     ///                {
148     ///                    var matchedLink = Memory.GetLinkValue(link);
149     ///                    var newValue = Memory.GetLinkValue(link);
150     ///                    newValue[Constants.IndexPart] = Constants.Itself;
151     ///                    newValue[Constants.SourcePart] =
152     ↳ Equals(substitution[Constants.SourcePart], Constants.Itself) ?
153     ↳ matchedLink[Constants.IndexPart] : substitution[Constants.SourcePart];
154     ///                    newValue[Constants.TargetPart] =
155     ↳ Equals(substitution[Constants.TargetPart], Constants.Itself) ?
156     ↳ matchedLink[Constants.IndexPart] : substitution[Constants.TargetPart];
157     ///                    var matchDecision = matchedHandler(matchedLink, newValue);
158     ///                    if (Equals(matchDecision, Constants.Break))
159     ///                    {
160     ///                        return false;
161     ///                    }
162     ///                    if (!Equals(matchDecision, Constants.Skip))
163     ///                    {
164     ///                        transitions.Add(new Transition(matchedLink, newValue));
165     ///                    }
166     ///                    return true;
167     ///                };
168     ///                if (!Memory.Each(handler, restriction))
169     ///                {
170     ///                    return Constants.Break;
171     ///                }
172     ///            }
173     ///        }
174     ///    }
175     ///    else
176     ///    {
177     ///        Func<T, bool> handler = link =>
178     ///        {
179     ///            var matchedLink = Memory.GetLinkValue(link);
180     ///            var matchDecision = matchedHandler(matchedLink, matchedLink);
181     ///            return !Equals(matchDecision, Constants.Break);
182     ///        };
183     ///        if (!Memory.Each(handler, restriction))
184     ///        {
185     ///            return Constants.Break;
186     ///        }
187     ///    }
188     ///    else
189     ///    {
190     ///        if (substitution != null)
191     ///        {
192     ///            transitions = new List<IList<T>>();
193     ///            Func<T, bool> handler = link =>
194     ///            {
195     ///                var matchedLink = Memory.GetLinkValue(link);
196     ///                var matchDecision = matchedHandler(matchedLink, matchedLink);
197     ///                return !Equals(matchDecision, Constants.Break);
198     ///            };
199     ///            if (!Memory.Each(handler, restriction))
200     ///            {
201     ///                return Constants.Break;
202     ///            }
203     ///        }
204     ///    }
205     ///    return transitions;
206 }

```

```

178         transitions.Add(matchedLink);
179         return true;
180     };
181     if (!Memory.Each(handler, restriction))
182         return Constants.Break;
183     }
184     else
185     {
186         return Constants.Continue;
187     }
188 }
189 }
190 if (substitution != null)
191 {
192     // Есть причина делать замену (запись)
193     if (substitutedHandler != null)
194     {
195     }
196     else
197     {
198     }
199 }
200 return Constants.Continue;
201
202 //if (restriction.IsNullOrEmpty()) // Create
203 //{
204 //    substitution[Constants.IndexPart] = Memory.AllocateLink();
205 //    Memory.SetLinkValue(substitution);
206 //}
207 //else if (substitution.IsNullOrEmpty()) // Delete
208 //{
209 //    Memory.FreeLink(restriction[Constants.IndexPart]);
210 //}
211 //else if (restriction.EqualTo(substitution)) // Read or ("repeat" the state) // Each
212 //{
213 //    // No need to collect links to list
214 //    // Skip == Continue
215 //    // No need to check substitutedHandler
216 //    if (!Memory.Each(link => !Equals(matchedHandler(Memory.GetLinkValue(link)),
217 //        ↪ Constants.Break), restriction))
218 //        return Constants.Break;
219 //}
220 //else // Update
221 //{
222 //    //List<IList<T>> matchedLinks = null;
223 //    if (matchedHandler != null)
224 //    {
225 //        matchedLinks = new List<IList<T>>();
226 //        Func<T, bool> handler = link =>
227 //        {
228 //            var matchedLink = Memory.GetLinkValue(link);
229 //            var matchDecision = matchedHandler(matchedLink);
230 //            if (Equals(matchDecision, Constants.Break))
231 //                return false;
232 //            if (!Equals(matchDecision, Constants.Skip))
233 //                matchedLinks.Add(matchedLink);
234 //            return true;
235 //        };
236 //        if (!Memory.Each(handler, restriction))
237 //            return Constants.Break;
238 //    }
239 //    if (!matchedLinks.IsNullOrEmpty())
240 //    {
241 //        var totalMatchedLinks = matchedLinks.Count;
242 //        for (var i = 0; i < totalMatchedLinks; i++)
243 //        {
244 //            var matchedLink = matchedLinks[i];
245 //            if (substitutedHandler != null)
246 //            {
247 //                var newValue = new List<T>(); // TODO: Prepare value to update here
248 //                // TODO: Decide is it actually needed to use Before and After
249 //                ↪ substitution handling.
250 //                var substitutedDecision = substitutedHandler(matchedLink,
251 //                ↪ newValue);
252 //                if (Equals(substitutedDecision, Constants.Break))
253 //                    return Constants.Break;
254 //                if (Equals(substitutedDecision, Constants.Continue))

```

```

252         //          {
253         //              // Actual update here
254         //              Memory.SetLinkValue(newValue);
255         //          }
256         //          if (Equals(substitutedDecision, Constants.Skip))
257         //          {
258         //              // Cancel the update. TODO: decide use separate Cancel
259         //              ↪ constant or Skip is enough?
260         //          }
261         //      }
262         //  }
263     ///}
264     return _constants.Continue;
265 }
266
267 /// <summary>
268 /// <para>
269 /// Triggers the pattern or condition.
270 /// </para>
271 /// <para></para>
272 /// </summary>
273 /// <param name="patternOrCondition">
274 /// <para>The pattern or condition.</para>
275 /// <para></para>
276 /// </param>
277 /// <param name="matchHandler">
278 /// <para>The match handler.</para>
279 /// <para></para>
280 /// </param>
281 /// <param name="substitution">
282 /// <para>The substitution.</para>
283 /// <para></para>
284 /// </param>
285 /// <param name="substitutionHandler">
286 /// <para>The substitution handler.</para>
287 /// <para></para>
288 /// </param>
289 /// <exception cref="NotImplementedException">
290 /// <para></para>
291 /// <para></para>
292 /// </exception>
293 /// <exception cref="NotSupportedException">
294 /// <para></para>
295 /// <para></para>
296 /// </exception>
297 /// <exception cref="NotSupportedException">
298 /// <para></para>
299 /// <para></para>
300 /// </exception>
301 /// <exception cref="NotSupportedException">
302 /// <para></para>
303 /// <para></para>
304 /// </exception>
305 /// <exception cref="NotSupportedException">
306 /// <para></para>
307 /// <para></para>
308 /// </exception>
309 /// <returns>
310 /// <para>The link</para>
311 /// <para></para>
312 /// </returns>
313 public TLink Trigger(IList<TLink> patternOrCondition, Func<IList<TLink>, TLink>
    ↪ matchHandler, IList<TLink> substitution, Func<IList<TLink>, IList<TLink>, TLink>
    ↪ substitutionHandler)
314 {
315     var constants = _constants;
316     if (patternOrCondition.IsNullOrEmpty() && substitution.IsNullOrEmpty())
317     {
318         return constants.Continue;
319     }
320     else if (patternOrCondition.EqualTo(substitution)) // Should be Each here TODO:
321     ↪ Check if it is a correct condition
322     {
323         // Or it only applies to trigger without matchHandler.
324         throw new NotImplementedException();
325     }
    else if (!substitution.IsNullOrEmpty()) // Creation

```

```

326 {
327     var before = Array.Empty<TLink>();
328     // Что должно означать False здесь? Остановиться (перестать идти) или пропустить
329     //   ↳ (пройти мимо) или пустить (взять)?
330     if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
331     //   ↳ constants.Break))
332     {
333         return constants.Break;
334     }
335     var after = (IList<TLink>)substitution.ToArray();
336     if (_equalityComparer.Equals(after[0], default))
337     {
338         var newLink = _links.Create();
339         after[0] = newLink;
340     }
341     if (substitution.Count == 1)
342     {
343         after = _links.GetLink(substitution[0]);
344     }
345     else if (substitution.Count == 3)
346     {
347         //Links.Create(after);
348     }
349     else
350     {
351         throw new NotSupportedException();
352     }
353     if (matchHandler != null)
354     {
355         return substitutionHandler(before, after);
356     }
357     return constants.Continue;
358 }
359 else if (!patternOrCondition.IsNullOrEmpty()) // Deletion
360 {
361     if (patternOrCondition.Count == 1)
362     {
363         var linkToDelete = patternOrCondition[0];
364         var before = _links.GetLink(linkToDelete);
365         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
366         //   ↳ constants.Break))
367         {
368             return constants.Break;
369         }
370         var after = Array.Empty<TLink>();
371         _links.Update(linkToDelete, constants.Null, constants.Null);
372         _links.Delete(linkToDelete);
373         if (matchHandler != null)
374         {
375             return substitutionHandler(before, after);
376         }
377         return constants.Continue;
378     }
379     else
380     {
381         throw new NotSupportedException();
382     }
383 }
384 else // Replace / Update
385 {
386     if (patternOrCondition.Count == 1) //-V3125
387     {
388         var linkToUpdate = patternOrCondition[0];
389         var before = _links.GetLink(linkToUpdate);
390         if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
391         //   ↳ constants.Break))
392         {
393             return constants.Break;
394         }
395         var after = (IList<TLink>)substitution.ToArray(); //-V3125
396         if (_equalityComparer.Equals(after[0], default))
397         {
398             after[0] = linkToUpdate;
399         }
400         if (substitution.Count == 1)
401         {
402             if (!_equalityComparer.Equals(substitution[0], linkToUpdate))
403             {

```

```

400         after = _links.GetLink(substitution[0]);
401         _links.Update(linkToUpdate, constants.Null, constants.Null);
402         _links.Delete(linkToUpdate);
403     }
404 }
405 else if (substitution.Count == 3)
406 {
407     //Links.Update(after);
408 }
409 else
410 {
411     throw new NotSupportedException();
412 }
413 if (matchHandler != null)
414 {
415     return substitutionHandler(before, after);
416 }
417 return constants.Continue;
418 }
419 else
420 {
421     throw new NotSupportedException();
422 }
423 }
424 }
425
426 /// <remarks>
427 /// IList[IList[IList[T]]]
428 /// | | | |
429 /// | | | |
430 /// | | | |
431 /// | | | |
432 /// | | | |
433 /// | | | |
434 /// | | | |
435 /// | | | |
436 public IList<IList<IList<TLink>>> Trigger(IList<TLink> condition, IList<TLink>
    ↳ substitution)
437 {
438     var changes = new List<IList<IList<TLink>>>();
439     var @continue = _constants.Continue;
440     Trigger(condition, AlwaysContinue, substitution, (before, after) =>
441     {
442         var change = new[] { before, after };
443         changes.Add(change);
444         return @continue;
445     });
446     return changes;
447 }
448
449 /// <summary>
450 /// <para>
451 /// Alwayses the continue using the specified link to match.
452 /// </para>
453 /// <para></para>
454 /// </summary>
455 /// <param name="linkToMatch">
456 /// <para>The link to match.</para>
457 /// <para></para>
458 /// </param>
459 /// <returns>
460 /// <para>The link</para>
461 /// <para></para>
462 /// </returns>
463 private TLink AlwaysContinue(IList<TLink> linkToMatch) => _constants.Continue;
464 }
465 }

```

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      /// <summary>
21      /// <para>
22      /// The default.
23      /// </para>
24      /// <para></para>
25      /// </summary>
26      private static readonly EqualityComparer<T> _equalityComparer =
27          ↳ EqualityComparer<T>.Default;
28      /// <summary>
29      /// <para>.</para>
30      /// <para>.</para>
31      /// </summary>
32      /// <typeparam name="T">
33      /// <para>.</para>
34      /// <para>.</para>
35      /// </typeparam>
36      public readonly T Source;
37
38      /// <summary>
39      /// <para>.</para>
40      /// <para>.</para>
41      /// </summary>
42      /// <typeparam name="T">
43      /// <para>.</para>
44      /// <para>.</para>
45      /// </typeparam>
46      public readonly T Target;
47
48      /// <summary>
49      /// <para>.</para>
50      /// <para>.</para>
51      /// </summary>
52      /// <typeparam name="T">
53      /// <para>.</para>
54      /// <para>.</para>
55      /// </typeparam>
56      /// <param name="source">
57      /// <para>.</para>
58      /// <para>.</para>
59      /// </param>
60      /// <param name="target">
61      /// <para>.</para>
62      /// <para>.</para>
63      /// </param>
64      [MethodImpl(MethodImplOptions.AggressiveInlining)]
65      public Doublet(T source, T target)
66      {
67          Source = source;
68          Target = target;
69      }
70
71      /// <summary>
72      /// <para>.</para>
73      /// <para>.</para>
74      /// </summary>
75      /// <returns>
76      /// <para>.</para>
77      /// <para>.</para>
78      /// </returns>
79      [MethodImpl(MethodImplOptions.AggressiveInlining)]
80      public override string ToString() => $"{Source}->{Target}";
81
82      /// <summary>
83      /// <para>.</para>
84      /// <para>.</para>
85      /// </summary>
86      /// <typeparam>
87      /// <para>.</para>

```

```

88     /// <para>.</para>
89     /// </typeparam>
90     /// <param name="other">
91     /// <para>.</para>
92     /// <para>.</para>
93     /// </param>
94     /// <returns>
95     /// <para>.</para>
96     /// <para>.</para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public bool Equals(Doublet<T> other) => _equalityComparer.Equals(Source, other.Source)
    ↳ && _equalityComparer.Equals(Target, other.Target);

100
101     /// <summury>
102     /// <para>.</para>
103     /// <para>.</para>
104     /// </summury>
105     /// <typeparam>
106     /// <para>.</para>
107     /// <para>.</para>
108     /// </typeparam>
109     /// <param name="obj">
110     /// <para>.</para>
111     /// <para>.</para>
112     /// </param>
113     /// <returns>
114     /// <para>.</para>
115     /// <para>.</para>
116     /// </returns>
117     [MethodImpl(MethodImplOptions.AggressiveInlining)]
118     public override bool Equals(object obj) => obj is Doublet<T> doublet ?
    ↳ base.Equals(doublet) : false;

119
120     /// <summury>
121     /// <para>.</para>
122     /// <para>.</para>
123     /// </summury>
124     /// <returns>
125     /// <para>.</para>
126     /// <para>.</para>
127     /// </returns>
128     [MethodImpl(MethodImplOptions.AggressiveInlining)]
129     public override int GetHashCode() => (Source, Target).GetHashCode();

130
131     /// <summury>
132     /// <para>.</para>
133     /// <para>.</para>
134     /// </summury>
135     /// <param name="left">
136     /// <para>.</para>
137     /// <para>.</para>
138     /// </param>
139     /// <param name="right">
140     /// <para>.</para>
141     /// <para>.</para>
142     /// </param>
143     /// <returns>
144     /// <para>.</para>
145     /// <para>.</para>
146     /// </returns>
147     [MethodImpl(MethodImplOptions.AggressiveInlining)]
148     public static bool operator ==(Doublet<T> left, Doublet<T> right) => left.Equals(right);

149
150     /// <summury>
151     /// <para>.</para>
152     /// <para>.</para>
153     /// </summury>
154     /// <param name="left">
155     /// <para>.</para>
156     /// <para>.</para>
157     /// </param>
158     /// <param name="right">
159     /// <para>.</para>
160     /// <para>.</para>
161     /// </param>
162     /// <returns>
163     /// <para>.</para>

```

```

164     /// <para>.</para>
165     /// </returns>
166     [MethodImpl(MethodImplOptions.AggressiveInlining)]
167     public static bool operator !=(Doublet<T> left, Doublet<T> right) => !(left == right);
168 }
169 }

```

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

```

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

```

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

```

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

```

```

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

```

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

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Runtime.CompilerServices;
6  using Platform.Ranges;
7  using Platform.Collections.Arrays;
8  using Platform.Random;
9  using Platform.Setters;
10 using Platform.Converters;
11 using Platform.Numbers;
12 using Platform.Data.Exceptions;
13 using Platform.Data.Doublets.Decorators;
14
15 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
16
17 namespace Platform.Data.Doublets
18 {
19     /// <summary>
20     /// <para>
21     /// Represents the links extensions.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     public static class ILinksExtensions
26     {
27         /// <summary>
28         /// <para>
29         /// Runs the random creations using the specified links.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <typeparam name="TLink">
34         /// <para>The link.</para>
35         /// <para></para>
36         /// </typeparam>
37         /// <param name="links">
38         /// <para>The links.</para>
39         /// <para></para>
40         /// </param>
41         /// <param name="amountOfCreations">
42         /// <para>The amount of creations.</para>
43         /// <para></para>
44         /// </param>
45         [MethodImpl(MethodImplOptions.AggressiveInlining)]
46         public static void RunRandomCreations<TLink>(this ILinks<TLink> links, ulong
47             ↪ amountOfCreations)
48         {
49             var random = RandomHelpers.Default;
50             var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
51             var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
52             for (var i = 0UL; i < amountOfCreations; i++)
53             {
54                 var linksAddressRange = new Range<ulong>(0,
55                     ↪ addressToUInt64Converter.Convert(links.Count()));
56                 var source =
57                     ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
58                 var target =
59                     ↪ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
60                 links.GetOrCreate(source, target);
61             }
62         }
63
64         /// <summary>
65         /// <para>
66         /// Runs the random searches using the specified links.
67         /// </para>
68         /// <para></para>
69         /// </summary>

```

```

65     /// </summary>
66     /// <typeparam name="TLink">
67     /// <para>The link.</para>
68     /// <para></para>
69     /// </typeparam>
70     /// <param name="links">
71     /// <para>The links.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="amountOfSearches">
75     /// <para>The amount of searches.</para>
76     /// <para></para>
77     /// </param>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     public static void RunRandomSearches<TLink>(this ILinks<TLink> links, ulong
        ↳ amountOfSearches)
80     {
81         var random = RandomHelpers.Default;
82         var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
83         var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
84         for (var i = 0UL; i < amountOfSearches; i++)
85         {
86             var linksAddressRange = new Range<ulong>(0,
        ↳ addressToUInt64Converter.Convert(links.Count()));
87             var source =
        ↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
88             var target =
        ↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
89             links.SearchOrDefault(source, target);
90         }
91     }
92
93     /// <summary>
94     /// <para>
95     /// Runs the random deletions using the specified links.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     /// <typeparam name="TLink">
100    /// <para>The link.</para>
101    /// <para></para>
102    /// </typeparam>
103    /// <param name="links">
104    /// <para>The links.</para>
105    /// <para></para>
106    /// </param>
107    /// <param name="amountOfDeletions">
108    /// <para>The amount of deletions.</para>
109    /// <para></para>
110    /// </param>
111    [MethodImpl(MethodImplOptions.AggressiveInlining)]
112    public static void RunRandomDeletions<TLink>(this ILinks<TLink> links, ulong
        ↳ amountOfDeletions)
113    {
114        var random = RandomHelpers.Default;
115        var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
116        var uint64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
117        var linksCount = addressToUInt64Converter.Convert(links.Count());
118        var min = amountOfDeletions > linksCount ? 0UL : linksCount - amountOfDeletions;
119        for (var i = 0UL; i < amountOfDeletions; i++)
120        {
121            linksCount = addressToUInt64Converter.Convert(links.Count());
122            if (linksCount <= min)
123            {
124                break;
125            }
126            var linksAddressRange = new Range<ulong>(min, linksCount);
127            var link =
        ↳ uint64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
128            links.Delete(link);
129        }
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         ↪ links.Delete(new LinkAddress<TLink>(linkToDelete));
153
154     /// <remarks>
155     /// TODO: Возможно есть очень простой способ это сделать.
156     /// (Например просто удалить файл, или изменить его размер таким образом,
157     /// чтобы удалился весь контент)
158     /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
159     /// </remarks>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public static void DeleteAll<TLink>(this ILinks<TLink> links)
162     {
163         var equalityComparer = EqualityComparer<TLink>.Default;
164         var comparer = Comparer<TLink>.Default;
165         for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
166             ↪ Arithmetic.Decrement(i))
167         {
168             links.Delete(i);
169             if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
170             {
171                 i = links.Count();
172             }
173         }
174     }
175
176     /// <summary>
177     /// <para>
178     /// Firsts the links.
179     /// </para>
180     /// <para></para>
181     /// </summary>
182     /// <typeparam name="TLink">
183     /// <para>The link.</para>
184     /// <para></para>
185     /// </typeparam>
186     /// <param name="links">
187     /// <para>The links.</para>
188     /// <para></para>
189     /// </param>
190     /// <exception cref="InvalidOperationException">
191     /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
192     /// <para></para>
193     /// </exception>
194     /// <exception cref="InvalidOperationException">
195     /// <para>В хранилище нет связей.</para>
196     /// <para></para>
197     /// </exception>
198     /// <returns>
199     /// <para>The first link.</para>
200     /// <para></para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     public static TLink First<TLink>(this ILinks<TLink> links)
204     {
205         TLink firstLink = default;
206         var equalityComparer = EqualityComparer<TLink>.Default;
207         if (equalityComparer.Equals(links.Count(), default))
208         {
209             throw new InvalidOperationException("В хранилище нет связей.");
210         }
211         links.Each(links.Constants.Any, links.Constants.Any, link =>
212         {
213             firstLink = link[links.Constants.IndexPart];
214             return links.Constants.Break;
215         });
216     }

```

```

213     });
214     if (equalityComparer.Equals(firstLink, default))
215     {
216         throw new InvalidOperationException("В процессе поиска по хранилищу не было
        ↳ найдено связей.");
217     }
218     return firstLink;
219 }
220
221 /// <summary>
222 /// <para>
223 /// Singles the or default using the specified links.
224 /// </para>
225 /// <para></para>
226 /// </summary>
227 /// <typeparam name="TLink">
228 /// <para>The link.</para>
229 /// <para></para>
230 /// </typeparam>
231 /// <param name="links">
232 /// <para>The links.</para>
233 /// <para></para>
234 /// </param>
235 /// <param name="query">
236 /// <para>The query.</para>
237 /// <para></para>
238 /// </param>
239 /// <returns>
240 /// <para>The result.</para>
241 /// <para></para>
242 /// </returns>
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 public static IList<TLink> SingleOrDefault<TLink>(this ILinks<TLink> links, IList<TLink>
    ↳ query)
245 {
246     IList<TLink> result = null;
247     var count = 0;
248     var constants = links.Constants;
249     var @continue = constants.Continue;
250     var @break = constants.Break;
251     links.Each(linkHandler, query);
252     return result;
253
254     TLink linkHandler(IList<TLink> link)
255     {
256         if (count == 0)
257         {
258             result = link;
259             count++;
260             return @continue;
261         }
262         else
263         {
264             result = null;
265             return @break;
266         }
267     }
268 }
269
270 #region Paths
271
272 /// <remarks>
273 /// TODO: Как так? Как то что ниже может быть корректно?
274 /// Скорее всего практически не применимо
275 /// Предполагалось, что можно было конвертировать формируемый в проходе через
    ↳ SequenceWalker
276 /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
277 /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
278 /// </remarks>
279 [MethodImpl(MethodImplOptions.AggressiveInlining)]
280 public static bool CheckPathExistance<TLink>(this ILinks<TLink> links, params TLink[]
    ↳ path)
281 {
282     var current = path[0];
283     //EnsureLinkExists(current, "path");
284     if (!links.Exists(current))
285     {
286         return false;
287     }

```

```

288 var equalityComparer = EqualityComparer<TLink>.Default;
289 var constants = links.Constants;
290 for (var i = 1; i < path.Length; i++)
291 {
292     var next = path[i];
293     var values = links.GetLink(current);
294     var source = values[constants.SourcePart];
295     var target = values[constants.TargetPart];
296     if (equalityComparer.Equals(source, target) && equalityComparer.Equals(source,
        ↪ next))
297     {
298         //throw new InvalidOperationException(string.Format("Невозможно выбрать
        ↪ путь, так как и Source и Target совпадают с элементом пути {0}.", next));
299         return false;
300     }
301     if (!equalityComparer.Equals(next, source) && !equalityComparer.Equals(next,
        ↪ target))
302     {
303         //throw new InvalidOperationException(string.Format("Невозможно продолжить
        ↪ путь через элемент пути {0}", next));
304         return false;
305     }
306     current = next;
307 }
308 return true;
309 }
310
311 /// <remarks>
312 /// Может потребовать дополнительного стека для PathElement's при использовании
    ↪ SequenceWalker.
313 /// </remarks>
314 [MethodImpl(MethodImplOptions.AggressiveInlining)]
315 public static TLink GetByKeyes<TLink>(this ILinks<TLink> links, TLink root, params int[]
    ↪ path)
316 {
317     links.EnsureLinkExists(root, "root");
318     var currentLink = root;
319     for (var i = 0; i < path.Length; i++)
320     {
321         currentLink = links.GetLink(currentLink)[path[i]];
322     }
323     return currentLink;
324 }
325
326 /// <summary>
327 /// <para>
328 /// Gets the square matrix sequence element by index using the specified links.
329 /// </para>
330 /// <para></para>
331 /// </summary>
332 /// <typeparam name="TLink">
333 /// <para>The link.</para>
334 /// <para></para>
335 /// </typeparam>
336 /// <param name="links">
337 /// <para>The links.</para>
338 /// <para></para>
339 /// </param>
340 /// <param name="root">
341 /// <para>The root.</para>
342 /// <para></para>
343 /// </param>
344 /// <param name="size">
345 /// <para>The size.</para>
346 /// <para></para>
347 /// </param>
348 /// <param name="index">
349 /// <para>The index.</para>
350 /// <para></para>
351 /// </param>
352 /// <exception cref="ArgumentOutOfRangeException">
353 /// <para>Sequences with sizes other than powers of two are not supported.</para>
354 /// <para></para>
355 /// </exception>
356 /// <returns>
357 /// <para>The current link.</para>
358 /// <para></para>
359 /// </returns>

```



```

360 [MethodImpl(MethodImplOptions.AggressiveInlining)]
361 public static TLink GetSquareMatrixSequenceElementByIndex<TLink>(this ILinks<TLink>
    ↳ links, TLink root, ulong size, ulong index)
362 {
363     var constants = links.Constants;
364     var source = constants.SourcePart;
365     var target = constants.TargetPart;
366     if (!Platform.Numbers.Math.IsPowerOfTwo(size))
367     {
368         throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
            ↳ than powers of two are not supported.");
369     }
370     var path = new BitArray(BitConverter.GetBytes(index));
371     var length = Bit.GetLowestPosition(size);
372     links.EnsureLinkExists(root, "root");
373     var currentLink = root;
374     for (var i = length - 1; i >= 0; i--)
375     {
376         currentLink = links.GetLink(currentLink)[path[i] ? target : source];
377     }
378     return currentLink;
379 }
380
381 #endregion
382
383 /// <summary>
384 /// Возвращает индекс указанной связи.
385 /// </summary>
386 /// <param name="links">Хранилище связей.</param>
387 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
388 /// <returns>Индекс начальной связи для указанной связи.</returns>
389 [MethodImpl(MethodImplOptions.AggressiveInlining)]
390 public static TLink GetIndex<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
    ↳ link[links.Constants.IndexPart];
391
392 /// <summary>
393 /// Возвращает индекс начальной (Source) связи для указанной связи.
394 /// </summary>
395 /// <param name="links">Хранилище связей.</param>
396 /// <param name="link">Индекс связи.</param>
397 /// <returns>Индекс начальной связи для указанной связи.</returns>
398 [MethodImpl(MethodImplOptions.AggressiveInlining)]
399 public static TLink GetSource<TLink>(this ILinks<TLink> links, TLink link) =>
    ↳ links.GetLink(link)[links.Constants.SourcePart];
400
401 /// <summary>
402 /// Возвращает индекс начальной (Source) связи для указанной связи.
403 /// </summary>
404 /// <param name="links">Хранилище связей.</param>
405 /// <param name="link">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
406 /// <returns>Индекс начальной связи для указанной связи.</returns>
407 [MethodImpl(MethodImplOptions.AggressiveInlining)]
408 public static TLink GetSource<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
    ↳ link[links.Constants.SourcePart];
409
410 /// <summary>
411 /// Возвращает индекс конечной (Target) связи для указанной связи.
412 /// </summary>
413 /// <param name="links">Хранилище связей.</param>
414 /// <param name="link">Индекс связи.</param>
415 /// <returns>Индекс конечной связи для указанной связи.</returns>
416 [MethodImpl(MethodImplOptions.AggressiveInlining)]
417 public static TLink GetTarget<TLink>(this ILinks<TLink> links, TLink link) =>
    ↳ links.GetLink(link)[links.Constants.TargetPart];
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">Связь представленная списком, состоящим из её адреса и
    ↳ содержимого.</param>
424 /// <returns>Индекс конечной связи для указанной связи.</returns>
425 [MethodImpl(MethodImplOptions.AggressiveInlining)]
426 public static TLink GetTarget<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
    ↳ link[links.Constants.TargetPart];
427

```

```

428 /// <summary>
429 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    → (handler) для каждой подходящей связи.
430 /// </summary>
431 /// <param name="links">Хранилище связей.</param>
432 /// <param name="handler">Обработчик каждой подходящей связи.</param>
433 /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
    → может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
    → Any - отсутствие ограничения, 1..∞ конкретный адрес связи.</param>
434 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    → случае.</returns>
435 [MethodImpl(MethodImplOptions.AggressiveInlining)]
436 public static bool Each<TLink>(this ILinks<TLink> links, Func<IList<TLink>, TLink>
    → handler, params TLink[] restrictions)
437     => EqualityComparer<TLink>.Default.Equals(links.Each(handler, restrictions),
    → links.Constants.Continue);

438 /// <summary>
439 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    → (handler) для каждой подходящей связи.
440 /// </summary>
441 /// <param name="links">Хранилище связей.</param>
442 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
    → (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
    → Constants.Any - любое начало, 1..∞ конкретное начало)</param>
443 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
    → (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
    → Constants.Any - любой конец, 1..∞ конкретный конец)</param>
444 /// <param name="handler">Обработчик каждой подходящей связи.</param>
445 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    → случае.</returns>
446 [MethodImpl(MethodImplOptions.AggressiveInlining)]
447 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
    → Func<TLink, bool> handler)
448 {
449     var constants = links.Constants;
450     return links.Each(link => handler(link[constants.IndexPart]) ? constants.Continue :
451         → constants.Break, constants.Any, source, target);
452 }

453 /// <summary>
454 /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
    → (handler) для каждой подходящей связи.
455 /// </summary>
456 /// <param name="links">Хранилище связей.</param>
457 /// <param name="source">Значение, определяющее соответствующие шаблону связи.
    → (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве начала,
    → Constants.Any - любое начало, 1..∞ конкретное начало)</param>
458 /// <param name="target">Значение, определяющее соответствующие шаблону связи.
    → (Constants.Null - 0-я связь, обозначающая ссылку на пустоту в качестве конца,
    → Constants.Any - любой конец, 1..∞ конкретный конец)</param>
459 /// <param name="handler">Обработчик каждой подходящей связи.</param>
460 /// <returns>True, в случае если проход по связям не был прерван и False в обратном
    → случае.</returns>
461 [MethodImpl(MethodImplOptions.AggressiveInlining)]
462 public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
    → Func<IList<TLink>, TLink> handler) => links.Each(handler, links.Constants.Any,
    → source, target);

463 /// <summary>
464 /// <para>
465 /// Alls the links.
466 /// </para>
467 /// <para></para>
468 /// </summary>
469 /// <typeparam name="TLink">
470 /// <para>The link.</para>
471 /// <para></para>
472 /// </typeparam>
473 /// <param name="links">
474 /// <para>The links.</para>
475 /// <para></para>
476 /// </param>
477 /// <param name="restrictions">
478 /// <para>The restrictions.</para>
479 /// <para></para>
480 /// </param>
481

```

```

482 /// </param>
483 /// <returns>
484 /// <para>A list of i list t link</para>
485 /// <para></para>
486 /// </returns>
487 [MethodImpl(MethodImplOptions.AggressiveInlining)]
488 public static IList<IList<TLink>> All<TLink>(this ILinks<TLink> links, params TLink[]
↳ restrictions)
489 {
490     var arraySize = CheckedConverter<TLink,
↳ ulong>.Default.Convert(links.Count(restrictions));
491     if (arraySize > 0)
492     {
493         var array = new IList<TLink>[arraySize];
494         var filler = new ArrayFiller<IList<TLink>, TLink>(array,
↳ links.Constants.Continue);
495         links.Each(filler.AddAndReturnConstant, restrictions);
496         return array;
497     }
498     else
499     {
500         return Array.Empty<IList<TLink>>();
501     }
502 }
503
504 /// <summary>
505 /// <para>
506 /// Alls the indices using the specified links.
507 /// </para>
508 /// <para></para>
509 /// </summary>
510 /// <typeparam name="TLink">
511 /// <para>The link.</para>
512 /// <para></para>
513 /// </typeparam>
514 /// <param name="links">
515 /// <para>The links.</para>
516 /// <para></para>
517 /// </param>
518 /// <param name="restrictions">
519 /// <para>The restrictions.</para>
520 /// <para></para>
521 /// </param>
522 /// <returns>
523 /// <para>A list of t link</para>
524 /// <para></para>
525 /// </returns>
526 [MethodImpl(MethodImplOptions.AggressiveInlining)]
527 public static IList<TLink> AllIndices<TLink>(this ILinks<TLink> links, params TLink[]
↳ restrictions)
528 {
529     var arraySize = CheckedConverter<TLink,
↳ ulong>.Default.Convert(links.Count(restrictions));
530     if (arraySize > 0)
531     {
532         var array = new TLink[arraySize];
533         var filler = new ArrayFiller<TLink, TLink>(array, links.Constants.Continue);
534         links.Each(filler.AddFirstAndReturnConstant, restrictions);
535         return array;
536     }
537     else
538     {
539         return Array.Empty<TLink>();
540     }
541 }
542
543 /// <summary>
544 /// Возвращает значение, определяющее существует ли связь с указанными началом и концом
↳ в хранилище связей.
545 /// </summary>
546 /// <param name="links">Хранилище связей.</param>
547 /// <param name="source">Начало связи.</param>
548 /// <param name="target">Конец связи.</param>
549 /// <returns>Значение, определяющее существует ли связь.</returns>
550 [MethodImpl(MethodImplOptions.AggressiveInlining)]
551 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;

```

```

552 #region Ensure
553 // TODO: May be move to EnsureExtensions or make it both there and here
554
555 /// <summary>
556 /// <para>
557 /// Ensures the link exists using the specified links.
558 /// </para>
559 /// <para></para>
560 /// </summary>
561 /// <typeparam name="TLink">
562 /// <para>The link.</para>
563 /// <para></para>
564 /// </typeparam>
565 /// <param name="links">
566 /// <para>The links.</para>
567 /// <para></para>
568 /// </param>
569 /// <param name="restrictions">
570 /// <para>The restrictions.</para>
571 /// <para></para>
572 /// </param>
573 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
574 /// <para>sequence[{i}]</para>
575 /// <para></para>
576 /// </exception>
577 [MethodImpl(MethodImplOptions.AggressiveInlining)]
578 public static void EnsureLinkExists<TLink>(this ILinks<TLink> links, IList<TLink>
579     ↪ restrictions)
580 {
581     for (var i = 0; i < restrictions.Count; i++)
582     {
583         if (!links.Exists(restrictions[i]))
584         {
585             throw new ArgumentLinkDoesNotExistsException<TLink>(restrictions[i],
586                 ↪ $"{sequence[{i}]"}");
587         }
588     }
589 }
590
591 /// <summary>
592 /// <para>
593 /// Ensures the inner reference exists using the specified links.
594 /// </para>
595 /// <para></para>
596 /// </summary>
597 /// <typeparam name="TLink">
598 /// <para>The link.</para>
599 /// <para></para>
600 /// </typeparam>
601 /// <param name="links">
602 /// <para>The links.</para>
603 /// <para></para>
604 /// </param>
605 /// <param name="reference">
606 /// <para>The reference.</para>
607 /// <para></para>
608 /// </param>
609 /// <param name="argumentName">
610 /// <para>The argument name.</para>
611 /// <para></para>
612 /// </param>
613 /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
614 /// <para></para>
615 /// <para></para>
616 /// </exception>
617 [MethodImpl(MethodImplOptions.AggressiveInlining)]
618 public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links, TLink
619     ↪ reference, string argumentName)
620 {
621     if (links.Constants.IsInternalReference(reference) && !links.Exists(reference))
622     {
623         throw new ArgumentLinkDoesNotExistsException<TLink>(reference, argumentName);
624     }
625 }
626
627 /// <summary>
628 /// <para>

```

```

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

```

```

700    /// <para></para>
701    /// </typeparam>
702    /// <param name="links">
703    /// <para>The links.</para>
704    /// <para></para>
705    /// </param>
706    /// <param name="link">
707    /// <para>The link.</para>
708    /// <para></para>
709    /// </param>
710    /// <param name="argumentName">
711    /// <para>The argument name.</para>
712    /// <para></para>
713    /// </param>
714    /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
715    /// <para></para>
716    /// <para></para>
717    /// </exception>
718    [MethodImpl(MethodImplOptions.AggressiveInlining)]
719    public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links, TLink link,
    ↪ string argumentName)
720    {
721        var equalityComparer = EqualityComparer<TLink>.Default;
722        if (!equalityComparer.Equals(link, links.Constants.Any) && !links.Exists(link))
723        {
724            throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
725        }
726    }
727
728    /// <summary>
729    /// <para>
730    /// Ensures the link is itself or exists using the specified links.
731    /// </para>
732    /// <para></para>
733    /// </summary>
734    /// <typeparam name="TLink">
735    /// <para>The link.</para>
736    /// <para></para>
737    /// </typeparam>
738    /// <param name="links">
739    /// <para>The links.</para>
740    /// <para></para>
741    /// </param>
742    /// <param name="link">
743    /// <para>The link.</para>
744    /// <para></para>
745    /// </param>
746    /// <param name="argumentName">
747    /// <para>The argument name.</para>
748    /// <para></para>
749    /// </param>
750    /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
751    /// <para></para>
752    /// <para></para>
753    /// </exception>
754    [MethodImpl(MethodImplOptions.AggressiveInlining)]
755    public static void EnsureLinkIsItselfOrExists<TLink>(this ILinks<TLink> links, TLink
    ↪ link, string argumentName)
756    {
757        var equalityComparer = EqualityComparer<TLink>.Default;
758        if (!equalityComparer.Equals(link, links.Constants.Itself) && !links.Exists(link))
759        {
760            throw new ArgumentLinkDoesNotExistsException<TLink>(link, argumentName);
761        }
762    }
763
764    /// <param name="links">Хранилище связей.</param>
765    [MethodImpl(MethodImplOptions.AggressiveInlining)]
766    public static void EnsureDoesNotExists<TLink>(this ILinks<TLink> links, TLink source,
    ↪ TLink target)
767    {
768        if (links.Exists(source, target))
769        {
770            throw new LinkWithSameValueAlreadyExistsException();
771        }
772    }
773
774    /// <param name="links">Хранилище связей.</param>

```

```

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

```

```

844 public static bool HasUsages<TLink>(this ILinks<TLink> links, TLink link) =>
845     ↳ Comparer<TLink>.Default.Compare(links.CountUsages(link), default) > 0;
846
847 /// <param name="links">Хранилище связей.</param>
848 [MethodImpl(MethodImplOptions.AggressiveInlining)]
849 public static bool Equals<TLink>(this ILinks<TLink> links, TLink link, TLink source,
850     ↳ TLink target)
851 {
852     var constants = links.Constants;
853     var values = links.GetLink(link);
854     var equalityComparer = EqualityComparer<TLink>.Default;
855     return equalityComparer.Equals(values[constants.SourcePart], source) &&
856         ↳ equalityComparer.Equals(values[constants.TargetPart], target);
857 }
858
859 /// <summary>
860 /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
861 /// </summary>
862 /// <param name="links">Хранилище связей.</param>
863 /// <param name="source">Индекс связи, которая является началом для искомой
864     ↳ связи.</param>
865 /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
866 /// <returns>Индекс искомой связи с указанными Source (началом) и Target
867     ↳ (концом).</returns>
868 [MethodImpl(MethodImplOptions.AggressiveInlining)]
869 public static TLink SearchOrDefault<TLink>(this ILinks<TLink> links, TLink source, TLink
870     ↳ target)
871 {
872     var constants = links.Constants;
873     var setter = new Setter<TLink, TLink>(constants.Continue, constants.Break, default);
874     links.Each(setter.SetFirstAndReturnFalse, constants.Any, source, target);
875     return setter.Result;
876 }
877
878 /// <param name="links">Хранилище связей.</param>
879 [MethodImpl(MethodImplOptions.AggressiveInlining)]
880 public static TLink Create<TLink>(this ILinks<TLink> links) => links.Create(null);
881
882 /// <param name="links">Хранилище связей.</param>
883 [MethodImpl(MethodImplOptions.AggressiveInlining)]
884 public static TLink CreatePoint<TLink>(this ILinks<TLink> links)
885 {
886     var link = links.Create();
887     return links.Update(link, link, link);
888 }
889
890 /// <param name="links">Хранилище связей.</param>
891 [MethodImpl(MethodImplOptions.AggressiveInlining)]
892 public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
893     ↳ target) => links.Update(links.Create(), source, target);
894
895 /// <summary>
896 /// Обновляет связь с указанными началом (Source) и концом (Target)
897 /// на связь с указанными началом (NewSource) и концом (NewTarget).
898 /// </summary>
899 /// <param name="links">Хранилище связей.</param>
900 /// <param name="link">Индекс обновляемой связи.</param>
901 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
902     ↳ выполняется обновление.</param>
903 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
904     ↳ выполняется обновление.</param>
905 /// <returns>Индекс обновлённой связи.</returns>
906 [MethodImpl(MethodImplOptions.AggressiveInlining)]
907 public static TLink Update<TLink>(this ILinks<TLink> links, TLink link, TLink newSource,
908     ↳ TLink newTarget) => links.Update(new LinkAddress<TLink>(link), new Link<TLink>(link,
909     ↳ newSource, newTarget));
910
911 /// <summary>
912 /// Обновляет связь с указанными началом (Source) и концом (Target)
913 /// на связь с указанными началом (NewSource) и концом (NewTarget).
914 /// </summary>
915 /// <param name="links">Хранилище связей.</param>
916 /// <param name="restrictions">Ограничения на содержимое связей. Каждое ограничение
917     ↳ может иметь значения: Constants.Null - 0-я связь, обозначающая ссылку на пустоту,
918     ↳ Itself - требование установить ссылку на себя, 1..∞ конкретный адрес другой
919     ↳ связи.</param>
920 /// <returns>Индекс обновлённой связи.</returns>
921 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

908 public static TLink Update<TLink>(this ILinks<TLink> links, params TLink[] restrictions)
909 {
910     if (restrictions.Length == 2)
911     {
912         return links.MergeAndDelete(restrictions[0], restrictions[1]);
913     }
914     if (restrictions.Length == 4)
915     {
916         return links.UpdateOrCreateOrGet(restrictions[0], restrictions[1],
917             ↪ restrictions[2], restrictions[3]);
918     }
919     else
920     {
921         return links.Update(new LinkAddress<TLink>(restrictions[0]), restrictions);
922     }
923 }
924
925 /// <summary>
926 /// <para>
927 /// Resolves the constant as self reference using the specified links.
928 /// </para>
929 /// <para></para>
930 /// </summary>
931 /// <typeparam name="TLink">
932 /// <para>The link.</para>
933 /// </typeparam>
934 /// <param name="links">
935 /// <para>The links.</para>
936 /// </param>
937 /// <param name="constant">
938 /// <para>The constant.</para>
939 /// </param>
940 /// <param name="restrictions">
941 /// <para>The restrictions.</para>
942 /// </param>
943 /// <param name="substitution">
944 /// <para>The substitution.</para>
945 /// </param>
946 /// <returns>
947 /// <para>A list of t link</para>
948 /// </returns>
949 [MethodImpl(MethodImplOptions.AggressiveInlining)]
950 public static IList<TLink> ResolveConstantAsSelfReference<TLink>(this ILinks<TLink>
951     ↪ links, TLink constant, IList<TLink> restrictions, IList<TLink> substitution)
952 {
953     var equalityComparer = EqualityComparer<TLink>.Default;
954     var constants = links.Constants;
955     var restrictionsIndex = restrictions[constants.IndexPart];
956     var substitutionIndex = substitution[constants.IndexPart];
957     if (equalityComparer.Equals(substitutionIndex, default))
958     {
959         substitutionIndex = restrictionsIndex;
960     }
961     var source = substitution[constants.SourcePart];
962     var target = substitution[constants.TargetPart];
963     source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;
964     target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
965     return new Link<TLink>(substitutionIndex, source, target);
966 }
967
968 /// <summary>
969 /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
970 ↪ с указанными Source (началом) и Target (концом).
971 /// </summary>
972 /// <param name="links">Хранилище связей.</param>
973 /// <param name="source">Индекс связи, которая является началом на создаваемой
974 ↪ связи.</param>
975 /// <param name="target">Индекс связи, которая является концом для создаваемой
976 ↪ связи.</param>
977 /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
978 [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

980 public static TLink GetOrCreate<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↪ target)
981 {
982     var link = links.SearchOrDefault(source, target);
983     if (EqualityComparer<TLink>.Default.Equals(link, default))
984     {
985         link = links.CreateAndUpdate(source, target);
986     }
987     return link;
988 }
989
990 /// <summary>
991 /// Обновляет связь с указанными началом (Source) и концом (Target)
992 /// на связь с указанными началом (NewSource) и концом (NewTarget).
993 /// </summary>
994 /// <param name="links">Хранилище связей.</param>
995 /// <param name="source">Индекс связи, которая является началом обновляемой
    ↪ связи.</param>
996 /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
997 /// <param name="newSource">Индекс связи, которая является началом связи, на которую
    ↪ выполняется обновление.</param>
998 /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
    ↪ выполняется обновление.</param>
999 /// <returns>Индекс обновлённой связи.</returns>
1000 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1001 public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
    ↪ TLink target, TLink newSource, TLink newTarget)
1002 {
1003     var equalityComparer = EqualityComparer<TLink>.Default;
1004     var link = links.SearchOrDefault(source, target);
1005     if (equalityComparer.Equals(link, default))
1006     {
1007         return links.CreateAndUpdate(newSource, newTarget);
1008     }
1009     if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
    ↪ target))
1010     {
1011         return link;
1012     }
1013     return links.Update(link, newSource, newTarget);
1014 }
1015
1016 /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1017 /// <param name="links">Хранилище связей.</param>
1018 /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
1019 /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
1020 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1021 public static TLink DeleteIfExists<TLink>(this ILinks<TLink> links, TLink source, TLink
    ↪ target)
1022 {
1023     var link = links.SearchOrDefault(source, target);
1024     if (!EqualityComparer<TLink>.Default.Equals(link, default))
1025     {
1026         links.Delete(link);
1027         return link;
1028     }
1029     return default;
1030 }
1031
1032 /// <summary>Удаляет несколько связей.</summary>
1033 /// <param name="links">Хранилище связей.</param>
1034 /// <param name="deletedLinks">Список адресов связей к удалению.</param>
1035 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1036 public static void DeleteMany<TLink>(this ILinks<TLink> links, IList<TLink> deletedLinks)
1037 {
1038     for (int i = 0; i < deletedLinks.Count; i++)
1039     {
1040         links.Delete(deletedLinks[i]);
1041     }
1042 }
1043
1044 /// <remarks>Before execution of this method ensure that deleted link is detached (all
    ↪ values - source and target are reset to null) or it might enter into infinite
    ↪ recursion.</remarks>
1045 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1046 public static void DeleteAllUsages<TLink>(this ILinks<TLink> links, TLink linkIndex)
1047 {
1048     var anyConstant = links.Constants.Any;

```

```

1049     var usagesAsSourceQuery = new Link<TLink>(anyConstant, linkIndex, anyConstant);
1050     links.DeleteByQuery(usagesAsSourceQuery);
1051     var usagesAsTargetQuery = new Link<TLink>(anyConstant, linkIndex, anyConstant);
1052     links.DeleteByQuery(usagesAsTargetQuery);
1053 }
1054
1055 /// <summary>
1056 /// <para>
1057 /// Deletes the by query using the specified links.
1058 /// </para>
1059 /// </summary>
1060 /// <typeparam name="TLink">
1061 /// <para>The link.</para>
1062 /// </typeparam>
1063 /// <param name="links">
1064 /// <para>The links.</para>
1065 /// </param>
1066 /// <param name="query">
1067 /// <para>The query.</para>
1068 /// </param>
1069 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1070 public static void DeleteByQuery<TLink>(this ILinks<TLink> links, Link<TLink> query)
1071 {
1072     var count = CheckedConverter<TLink, long>.Default.Convert(links.Count(query));
1073     if (count > 0)
1074     {
1075         var queryResult = new TLink[count];
1076         var queryResultFiller = new ArrayFiller<TLink, TLink>(queryResult,
1077             ↪ links.Constants.Continue);
1078         links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
1079         for (var i = count - 1; i >= 0; i--)
1080         {
1081             links.Delete(queryResult[i]);
1082         }
1083     }
1084 }
1085
1086 // TODO: Move to Platform.Data
1087 /// <summary>
1088 /// <para>
1089 /// Determines whether are values reset.
1090 /// </para>
1091 /// </summary>
1092 /// <typeparam name="TLink">
1093 /// <para>The link.</para>
1094 /// </typeparam>
1095 /// <param name="links">
1096 /// <para>The links.</para>
1097 /// </param>
1098 /// <param name="linkIndex">
1099 /// <para>The link index.</para>
1100 /// </param>
1101 /// <returns>
1102 /// <para>The bool</para>
1103 /// </returns>
1104 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1105 public static bool AreValuesReset<TLink>(this ILinks<TLink> links, TLink linkIndex)
1106 {
1107     var nullConstant = links.Constants.Null;
1108     var equalityComparer = EqualityComparer<TLink>.Default;
1109     var link = links.GetLink(linkIndex);
1110     for (int i = 1; i < link.Count; i++)
1111     {
1112         if (!equalityComparer.Equals(link[i], nullConstant))
1113         {
1114             return false;
1115         }
1116     }
1117     return true;
1118 }

```

```

1126 }
1127
1128 // TODO: Create a universal version of this method in Platform.Data (with using of for
    → loop)
1129 /// <summary>
1130 /// <para>
1131 /// Resets the values using the specified links.
1132 /// </para>
1133 /// <para></para>
1134 /// </summary>
1135 /// <typeparam name="TLink">
1136 /// <para>The link.</para>
1137 /// <para></para>
1138 /// </typeparam>
1139 /// <param name="links">
1140 /// <para>The links.</para>
1141 /// <para></para>
1142 /// </param>
1143 /// <param name="linkIndex">
1144 /// <para>The link index.</para>
1145 /// <para></para>
1146 /// </param>
1147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1148 public static void ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1149 {
1150     var nullConstant = links.Constants.Null;
1151     var updateRequest = new Link<TLink>(linkIndex, nullConstant, nullConstant);
1152     links.Update(updateRequest);
1153 }
1154
1155 // TODO: Create a universal version of this method in Platform.Data (with using of for
    → loop)
1156 /// <summary>
1157 /// <para>
1158 /// Enforces the reset values using the specified links.
1159 /// </para>
1160 /// <para></para>
1161 /// </summary>
1162 /// <typeparam name="TLink">
1163 /// <para>The link.</para>
1164 /// <para></para>
1165 /// </typeparam>
1166 /// <param name="links">
1167 /// <para>The links.</para>
1168 /// <para></para>
1169 /// </param>
1170 /// <param name="linkIndex">
1171 /// <para>The link index.</para>
1172 /// <para></para>
1173 /// </param>
1174 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1175 public static void EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
1176 {
1177     if (!links.AreValuesReset(linkIndex))
1178     {
1179         links.ResetValues(linkIndex);
1180     }
1181 }
1182
1183 /// <summary>
1184 /// Merging two usages graphs, all children of old link moved to be children of new link
    → or deleted.
1185 /// </summary>
1186 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1187 public static TLink MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
    → TLink newLinkIndex)
1188 {
1189     var addressToInt64Converter = CheckedConverter<TLink, long>.Default;
1190     var equalityComparer = EqualityComparer<TLink>.Default;
1191     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1192     {
1193         var constants = links.Constants;
1194         var usagesAsSourceQuery = new Link<TLink>(constants.Any, oldLinkIndex,
            → constants.Any);
1195         var usagesAsSourceCount =
            → addressToInt64Converter.Convert(links.Count(usagesAsSourceQuery));
1196         var usagesAsTargetQuery = new Link<TLink>(constants.Any, constants.Any,
            → oldLinkIndex);

```

```

1197     var usagesAsTargetCount =
1198         ↪ addressToInt64Converter.Convert(links.Count(usagesAsTargetQuery));
1199     var isStandalonePoint = Point<TLink>.IsFullPoint(links.GetLink(oldLinkIndex)) &&
1200         ↪ usagesAsSourceCount == 1 && usagesAsTargetCount == 1;
1201     if (!isStandalonePoint)
1202     {
1203         var totalUsages = usagesAsSourceCount + usagesAsTargetCount;
1204         if (totalUsages > 0)
1205         {
1206             var usages = ArrayPool.Allocate<TLink>(totalUsages);
1207             var usagesFiller = new ArrayFiller<TLink, TLink>(usages,
1208                 ↪ links.Constants.Continue);
1209             var i = 0L;
1210             if (usagesAsSourceCount > 0)
1211             {
1212                 links.Each(usagesFiller.AddFirstAndReturnConstant,
1213                     ↪ usagesAsSourceQuery);
1214                 for (; i < usagesAsSourceCount; i++)
1215                 {
1216                     var usage = usages[i];
1217                     if (!equalityComparer.Equals(usage, oldLinkIndex))
1218                     {
1219                         links.Update(usage, newLinkIndex, links.GetTarget(usage));
1220                     }
1221                 }
1222             }
1223             if (usagesAsTargetCount > 0)
1224             {
1225                 links.Each(usagesFiller.AddFirstAndReturnConstant,
1226                     ↪ usagesAsTargetQuery);
1227                 for (; i < usages.Length; i++)
1228                 {
1229                     var usage = usages[i];
1230                     if (!equalityComparer.Equals(usage, oldLinkIndex))
1231                     {
1232                         links.Update(usage, links.GetSource(usage), newLinkIndex);
1233                     }
1234                 }
1235             }
1236             ArrayPool.Free(usages);
1237         }
1238     }
1239     return newLinkIndex;
1240 }
1241
1242 /// <summary>
1243 /// Replace one link with another (replaced link is deleted, children are updated or
1244 /// ↪ deleted).
1245 /// </summary>
1246 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1247 public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
1248     ↪ TLink newLinkIndex)
1249 {
1250     var equalityComparer = EqualityComparer<TLink>.Default;
1251     if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
1252     {
1253         links.MergeUsages(oldLinkIndex, newLinkIndex);
1254         links.Delete(oldLinkIndex);
1255     }
1256     return newLinkIndex;
1257 }
1258
1259 /// <summary>
1260 /// <para>
1261 /// Decorates the with automatic uniqueness and usages resolution using the specified
1262 /// ↪ links.
1263 /// </para>
1264 /// <para></para>
1265 /// </summary>
1266 /// <typeparam name="TLink">
1267 /// <para>The link.</para>
1268 /// <para></para>
1269 /// </typeparam>
1270 /// <param name="links">
1271 /// <para>The links.</para>
1272 /// <para></para>
1273 /// </param>

```

```

1267     /// <returns>
1268     /// <para>The links.</para>
1269     /// <para></para>
1270     /// </returns>
1271     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1272     public static ILinks<TLink>
1273     ↪ DecorateWithAutomaticUniquenessAndUsagesResolution<TLink>(this ILinks<TLink> links)
1274     {
1275         links = new LinksCascadeUsagesResolver<TLink>(links);
1276         links = new NonNullContentsLinkDeletionResolver<TLink>(links);
1277         links = new LinksCascadeUniquenessAndUsagesResolver<TLink>(links);
1278         return links;
1279     }
1280     /// <summary>
1281     /// <para>
1282     /// Formats the links.
1283     /// </para>
1284     /// <para></para>
1285     /// </summary>
1286     /// <typeparam name="TLink">
1287     /// <para>The link.</para>
1288     /// <para></para>
1289     /// </typeparam>
1290     /// <param name="links">
1291     /// <para>The links.</para>
1292     /// <para></para>
1293     /// </param>
1294     /// <param name="link">
1295     /// <para>The link.</para>
1296     /// <para></para>
1297     /// </param>
1298     /// <returns>
1299     /// <para>The string</para>
1300     /// <para></para>
1301     /// </returns>
1302     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1303     public static string Format<TLink>(this ILinks<TLink> links, IList<TLink> link)
1304     {
1305         var constants = links.Constants;
1306         return $"({link[constants.IndexPart]}: {link[constants.SourcePart]})
1307         ↪ {link[constants.TargetPart]}";
1308     }
1309     /// <summary>
1310     /// <para>
1311     /// Formats the links.
1312     /// </para>
1313     /// <para></para>
1314     /// </summary>
1315     /// <typeparam name="TLink">
1316     /// <para>The link.</para>
1317     /// <para></para>
1318     /// </typeparam>
1319     /// <param name="links">
1320     /// <para>The links.</para>
1321     /// <para></para>
1322     /// </param>
1323     /// <param name="link">
1324     /// <para>The link.</para>
1325     /// <para></para>
1326     /// </param>
1327     /// <returns>
1328     /// <para>The string</para>
1329     /// <para></para>
1330     /// </returns>
1331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1332     public static string Format<TLink>(this ILinks<TLink> links, TLink link) =>
1333     ↪ links.Format(links.GetLink(link));
1334 }

```

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

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets
4  {
5      /// <summary>

```

```

6     /// <para>
7     /// Defines the synchronized links.
8     /// </para>
9     /// <para></para>
10    /// </summary>
11    /// <seealso cref="ISynchronizedLinks{TLink, ILinks{TLink}, LinksConstants{TLink}}"/>
12    /// <seealso cref="ILinks{TLink}"/>
13    public interface ISynchronizedLinks<TLink> : ISynchronizedLinks<TLink, ILinks<TLink>,
14        ↳ LinksConstants<TLink>>, ILinks<TLink>
15    {
16    }

```

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

```

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

```

```

64     /// <summary>
65     /// <para>
66     /// The target.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     public readonly TLink Target;
71
72     /// <summary>
73     /// <para>
74     /// Initializes a new <see cref="Link"/> instance.
75     /// </para>
76     /// <para></para>
77     /// </summary>
78     /// <param name="values">
79     /// <para>A values.</para>
80     /// <para></para>
81     /// </param>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     public Link(params TLink[] values) => SetValues(values, out Index, out Source, out
        ↪ Target);
84
85     /// <summary>
86     /// <para>
87     /// Initializes a new <see cref="Link"/> instance.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="values">
92     /// <para>A values.</para>
93     /// <para></para>
94     /// </param>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public Link(ICollection<TLink> values) => SetValues(values, out Index, out Source, out Target);
97
98     /// <summary>
99     /// <para>
100    /// Initializes a new <see cref="Link"/> instance.
101    /// </para>
102    /// <para></para>
103    /// </summary>
104    /// <param name="other">
105    /// <para>A other.</para>
106    /// <para></para>
107    /// </param>
108    /// <exception cref="NotSupportedException">
109    /// <para></para>
110    /// <para></para>
111    /// </exception>
112    [MethodImpl(MethodImplOptions.AggressiveInlining)]
113    public Link(object other)
114    {
115        if (other is Link<TLink> otherLink)
116        {
117            SetValues(ref otherLink, out Index, out Source, out Target);
118        }
119        else if (other is ICollection<TLink> otherList)
120        {
121            SetValues(otherList, out Index, out Source, out Target);
122        }
123        else
124        {
125            throw new NotSupportedException();
126        }
127    }
128
129    /// <summary>
130    /// <para>
131    /// Initializes a new <see cref="Link"/> instance.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="other">
136    /// <para>A other.</para>
137    /// <para></para>
138    /// </param>
139    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

140 public Link(ref Link<TLink> other) => SetValues(ref other, out Index, out Source, out
    ↳ Target);
141
142 /// <summary>
143 /// <para>
144 /// Initializes a new <see cref="Link"/> instance.
145 /// </para>
146 /// <para></para>
147 /// </summary>
148 /// <param name="index">
149 /// <para>A index.</para>
150 /// <para></para>
151 /// </param>
152 /// <param name="source">
153 /// <para>A source.</para>
154 /// <para></para>
155 /// </param>
156 /// <param name="target">
157 /// <para>A target.</para>
158 /// <para></para>
159 /// </param>
160 [MethodImpl(MethodImplOptions.AggressiveInlining)]
161 public Link(TLink index, TLink source, TLink target)
162 {
163     Index = index;
164     Source = source;
165     Target = target;
166 }
167
168 /// <summary>
169 /// <para>
170 /// Sets the values using the specified other.
171 /// </para>
172 /// <para></para>
173 /// </summary>
174 /// <param name="other">
175 /// <para>The other.</para>
176 /// <para></para>
177 /// </param>
178 /// <param name="index">
179 /// <para>The index.</para>
180 /// <para></para>
181 /// </param>
182 /// <param name="source">
183 /// <para>The source.</para>
184 /// <para></para>
185 /// </param>
186 /// <param name="target">
187 /// <para>The target.</para>
188 /// <para></para>
189 /// </param>
190 [MethodImpl(MethodImplOptions.AggressiveInlining)]
191 private static void SetValues(ref Link<TLink> other, out TLink index, out TLink source,
    ↳ out TLink target)
192 {
193     index = other.Index;
194     source = other.Source;
195     target = other.Target;
196 }
197
198 /// <summary>
199 /// <para>
200 /// Sets the values using the specified values.
201 /// </para>
202 /// <para></para>
203 /// </summary>
204 /// <param name="values">
205 /// <para>The values.</para>
206 /// <para></para>
207 /// </param>
208 /// <param name="index">
209 /// <para>The index.</para>
210 /// <para></para>
211 /// </param>
212 /// <param name="source">
213 /// <para>The source.</para>
214 /// <para></para>
215 /// </param>

```

```

216 /// <param name="target">
217 /// <para>The target.</para>
218 /// <para></para>
219 /// </param>
220 [MethodImpl(MethodImplOptions.AggressiveInlining)]
221 private static void SetValues(ICollection<TLink> values, out TLink index, out TLink source,
    ↪ out TLink target)
222 {
223     switch (values.Count)
224     {
225         case 3:
226             index = values[0];
227             source = values[1];
228             target = values[2];
229             break;
230         case 2:
231             index = values[0];
232             source = values[1];
233             target = default;
234             break;
235         case 1:
236             index = values[0];
237             source = default;
238             target = default;
239             break;
240         default:
241             index = default;
242             source = default;
243             target = default;
244             break;
245     }
246 }
247
248 /// <summary>
249 /// <para>
250 /// Gets the hash code.
251 /// </para>
252 /// <para></para>
253 /// </summary>
254 /// <returns>
255 /// <para>The int</para>
256 /// <para></para>
257 /// </returns>
258 [MethodImpl(MethodImplOptions.AggressiveInlining)]
259 public override int GetHashCode() => (Index, Source, Target).GetHashCode();
260
261 /// <summary>
262 /// <para>
263 /// Determines whether this instance is null.
264 /// </para>
265 /// <para></para>
266 /// </summary>
267 /// <returns>
268 /// <para>The bool</para>
269 /// <para></para>
270 /// </returns>
271 [MethodImpl(MethodImplOptions.AggressiveInlining)]
272 public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
273     && _equalityComparer.Equals(Source, _constants.Null)
274     && _equalityComparer.Equals(Target, _constants.Null);
275
276 /// <summary>
277 /// <para>
278 /// Determines whether this instance equals.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 /// <param name="other">
283 /// <para>The other.</para>
284 /// <para></para>
285 /// </param>
286 /// <returns>
287 /// <para>The bool</para>
288 /// <para></para>
289 /// </returns>
290 [MethodImpl(MethodImplOptions.AggressiveInlining)]
291 public override bool Equals(object other) => other is Link<TLink> &&
    ↪ Equals((Link<TLink>)other);
292

```

```

293     /// <summary>
294     /// <para>
295     /// Determines whether this instance equals.
296     /// </para>
297     /// <para></para>
298     /// </summary>
299     /// <param name="other">
300     /// <para>The other.</para>
301     /// <para></para>
302     /// </param>
303     /// <returns>
304     /// <para>The bool</para>
305     /// <para></para>
306     /// </returns>
307     [MethodImpl(MethodImplOptions.AggressiveInlining)]
308     public bool Equals(Link<TLink> other) => _equalityComparer.Equals(Index, other.Index)
309                                     && _equalityComparer.Equals(Source, other.Source)
310                                     && _equalityComparer.Equals(Target, other.Target);
311
312     /// <summary>
313     /// <para>
314     /// Returns the string using the specified index.
315     /// </para>
316     /// <para></para>
317     /// </summary>
318     /// <param name="index">
319     /// <para>The index.</para>
320     /// <para></para>
321     /// </param>
322     /// <param name="source">
323     /// <para>The source.</para>
324     /// <para></para>
325     /// </param>
326     /// <param name="target">
327     /// <para>The target.</para>
328     /// <para></para>
329     /// </param>
330     /// <returns>
331     /// <para>The string</para>
332     /// <para></para>
333     /// </returns>
334     [MethodImpl(MethodImplOptions.AggressiveInlining)]
335     public static string ToString(TLink index, TLink source, TLink target) => $"{({index}:
    ↳ {source}->{target})}";
336
337     /// <summary>
338     /// <para>
339     /// Returns the string using the specified source.
340     /// </para>
341     /// <para></para>
342     /// </summary>
343     /// <param name="source">
344     /// <para>The source.</para>
345     /// <para></para>
346     /// </param>
347     /// <param name="target">
348     /// <para>The target.</para>
349     /// <para></para>
350     /// </param>
351     /// <returns>
352     /// <para>The string</para>
353     /// <para></para>
354     /// </returns>
355     [MethodImpl(MethodImplOptions.AggressiveInlining)]
356     public static string ToString(TLink source, TLink target) => $"{({source}->{target})}";
357
358     [MethodImpl(MethodImplOptions.AggressiveInlining)]
359     public static implicit operator TLink[] (Link<TLink> link) => link.ToArray();
360
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     public static implicit operator Link<TLink>(TLink[] linkArray) => new
    ↳ Link<TLink>(linkArray);
363
364     /// <summary>
365     /// <para>
366     /// Returns the string.
367     /// </para>
368     /// <para></para>

```

```

369     /// </summary>
370     /// <returns>
371     /// <para>The string</para>
372     /// <para></para>
373     /// </returns>
374     [MethodImpl(MethodImplOptions.AggressiveInlining)]
375     public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
        ↳ ToString(Source, Target) : ToString(Index, Source, Target);

376
377     #region IList
378
379     /// <summary>
380     /// <para>
381     /// Gets the count value.
382     /// </para>
383     /// <para></para>
384     /// </summary>
385     public int Count
386     {
387         [MethodImpl(MethodImplOptions.AggressiveInlining)]
388         get => Length;
389     }
390
391     /// <summary>
392     /// <para>
393     /// Gets the is read only value.
394     /// </para>
395     /// <para></para>
396     /// </summary>
397     public bool IsReadOnly
398     {
399         [MethodImpl(MethodImplOptions.AggressiveInlining)]
400         get => true;
401     }
402
403     /// <summary>
404     /// <para>
405     /// The not supported exception.
406     /// </para>
407     /// <para></para>
408     /// </summary>
409     public TLink this[int index]
410     {
411         [MethodImpl(MethodImplOptions.AggressiveInlining)]
412         get
413         {
414             Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
415                 ↳ nameof(index));
416             if (index == _constants.IndexPart)
417             {
418                 return Index;
419             }
420             if (index == _constants.SourcePart)
421             {
422                 return Source;
423             }
424             if (index == _constants.TargetPart)
425             {
426                 return Target;
427             }
428             throw new NotSupportedException(); // Impossible path due to
429                 ↳ Ensure.ArgumentInRange
430         }
431         [MethodImpl(MethodImplOptions.AggressiveInlining)]
432         set => throw new NotSupportedException();
433     }
434
435     /// <summary>
436     /// <para>
437     /// Gets the enumerator.
438     /// </para>
439     /// <para></para>
440     /// </summary>
441     /// <returns>
442     /// <para>The enumerator</para>
443     /// <para></para>
444     /// </returns>
445     [MethodImpl(MethodImplOptions.AggressiveInlining)]
446     IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();

```

```

445     /// <summary>
446     /// <para>
447     /// Gets the enumerator.
448     /// </para>
449     /// <para></para>
450     /// </summary>
451     /// <returns>
452     /// <para>An enumerator of t link</para>
453     /// <para></para>
454     /// </returns>
455     [MethodImpl(MethodImplOptions.AggressiveInlining)]
456     public IEnumerator<TLink> GetEnumerator()
457     {
458         yield return Index;
459         yield return Source;
460         yield return Target;
461     }
462
463     /// <summary>
464     /// <para>
465     /// Adds the item.
466     /// </para>
467     /// <para></para>
468     /// </summary>
469     /// <param name="item">
470     /// <para>The item.</para>
471     /// <para></para>
472     /// </param>
473     [MethodImpl(MethodImplOptions.AggressiveInlining)]
474     public void Add(TLink item) => throw new NotSupportedException();
475
476     /// <summary>
477     /// <para>
478     /// Clears this instance.
479     /// </para>
480     /// <para></para>
481     /// </summary>
482     [MethodImpl(MethodImplOptions.AggressiveInlining)]
483     public void Clear() => throw new NotSupportedException();
484
485     /// <summary>
486     /// <para>
487     /// Determines whether this instance contains.
488     /// </para>
489     /// <para></para>
490     /// </summary>
491     /// <param name="item">
492     /// <para>The item.</para>
493     /// <para></para>
494     /// </param>
495     /// <returns>
496     /// <para>The bool</para>
497     /// <para></para>
498     /// </returns>
499     [MethodImpl(MethodImplOptions.AggressiveInlining)]
500     public bool Contains(TLink item) => IndexOf(item) >= 0;
501
502     /// <summary>
503     /// <para>
504     /// Copies the to using the specified array.
505     /// </para>
506     /// <para></para>
507     /// </summary>
508     /// <param name="array">
509     /// <para>The array.</para>
510     /// <para></para>
511     /// </param>
512     /// <param name="arrayIndex">
513     /// <para>The array index.</para>
514     /// <para></para>
515     /// </param>
516     /// <exception cref="InvalidOperationException">
517     /// <para></para>
518     /// <para></para>
519     /// </exception>
520     [MethodImpl(MethodImplOptions.AggressiveInlining)]
521     public void CopyTo(TLink[] array, int arrayIndex)

```

```

523 {
524     Ensure.OnDebug.ArgumentNotNull(array, nameof(array));
525     Ensure.OnDebug.ArgumentInRange(arrayIndex, new Range<int>(0, array.Length - 1),
        ↳ nameof(arrayIndex));
526     if (arrayIndex + Length > array.Length)
527     {
528         throw new InvalidOperationException();
529     }
530     array[arrayIndex++] = Index;
531     array[arrayIndex++] = Source;
532     array[arrayIndex] = Target;
533 }
534
535 /// <summary>
536 /// <para>
537 /// Determines whether this instance remove.
538 /// </para>
539 /// <para></para>
540 /// </summary>
541 /// <param name="item">
542 /// <para>The item.</para>
543 /// <para></para>
544 /// </param>
545 /// <returns>
546 /// <para>The bool</para>
547 /// <para></para>
548 /// </returns>
549 [MethodImpl(MethodImplOptions.AggressiveInlining)]
550 public bool Remove(TLink item) => Throw.A.NotSupportedExceptionAndReturn<bool>();
551
552 /// <summary>
553 /// <para>
554 /// Indexes the of using the specified item.
555 /// </para>
556 /// <para></para>
557 /// </summary>
558 /// <param name="item">
559 /// <para>The item.</para>
560 /// <para></para>
561 /// </param>
562 /// <returns>
563 /// <para>The int</para>
564 /// <para></para>
565 /// </returns>
566 [MethodImpl(MethodImplOptions.AggressiveInlining)]
567 public int IndexOf(TLink item)
568 {
569     if (_equalityComparer.Equals(Index, item))
570     {
571         return _constants.IndexPart;
572     }
573     if (_equalityComparer.Equals(Source, item))
574     {
575         return _constants.SourcePart;
576     }
577     if (_equalityComparer.Equals(Target, item))
578     {
579         return _constants.TargetPart;
580     }
581     return -1;
582 }
583
584 /// <summary>
585 /// <para>
586 /// Inserts the index.
587 /// </para>
588 /// <para></para>
589 /// </summary>
590 /// <param name="index">
591 /// <para>The index.</para>
592 /// <para></para>
593 /// </param>
594 /// <param name="item">
595 /// <para>The item.</para>
596 /// <para></para>
597 /// </param>
598 [MethodImpl(MethodImplOptions.AggressiveInlining)]
599 public void Insert(int index, TLink item) => throw new NotSupportedException();

```

```

600
601     /// <summary>
602     /// <para>
603     /// Removes the at using the specified index.
604     /// </para>
605     /// <para></para>
606     /// </summary>
607     /// <param name="index">
608     /// <para>The index.</para>
609     /// <para></para>
610     /// </param>
611     [MethodImpl(MethodImplOptions.AggressiveInlining)]
612     public void RemoveAt(int index) => throw new NotSupportedException();
613
614     [MethodImpl(MethodImplOptions.AggressiveInlining)]
615     public static bool operator ==(Link<TLink> left, Link<TLink> right) =>
616         ↪ left.Equals(right);
617
618     [MethodImpl(MethodImplOptions.AggressiveInlining)]
619     public static bool operator !=(Link<TLink> left, Link<TLink> right) => !(left == right);
620
621     #endregion
622 }

```

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     }
56 }

```

```

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

```

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

```

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

```

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

```

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

```



```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

```

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

```

```

133     /// <summary>
134     /// <para>
135     /// Gets the hash code.
136     /// </para>
137     /// <para></para>
138     /// </summary>
139     /// <returns>
140     /// <para>The int</para>
141     /// <para></para>
142     /// </returns>
143     [MethodImpl(MethodImplOptions.AggressiveInlining)]
144     public override int GetHashCode() => (AllocatedLinks, ReservedLinks, FreeLinks,
        ↪ FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();

145     [MethodImpl(MethodImplOptions.AggressiveInlining)]
146     public static bool operator ==(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
147         ↪ left.Equals(right);

148     [MethodImpl(MethodImplOptions.AggressiveInlining)]
149     public static bool operator !=(LinksHeader<TLink> left, LinksHeader<TLink> right) =>
150         ↪ !(left == right);
151 }
152 }

```

1.29 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethod

```

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

```

```

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

```

```

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

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

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

199
200    /// <summary>
201    /// <para>
202    /// Gets the link data part reference using the specified link.
203    /// </para>

```

```

204    /// <para></para>
205    /// </summary>
206    /// <param name="link">
207    /// <para>The link.</para>
208    /// <para></para>
209    /// </param>
210    /// <returns>
211    /// <para>A ref raw link data part of t link</para>
212    /// <para></para>
213    /// </returns>
214    [MethodImpl(MethodImplOptions.AggressiveInlining)]
215    protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↪ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));

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

232    /// <summary>
233    /// <para>
234    /// Gets the link values using the specified link index.
235    /// </para>
236    /// <para></para>
237    /// </summary>
238    /// <param name="linkIndex">
239    /// <para>The link index.</para>
240    /// <para></para>
241    /// </param>
242    /// <returns>
243    /// <para>A list of t link</para>
244    /// <para></para>
245    /// </returns>
246    [MethodImpl(MethodImplOptions.AggressiveInlining)]
247    protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
248    {
249        ref var link = ref GetLinkDataPartReference(linkIndex);
250        return new Link<TLink>(linkIndex, link.Source, link.Target);
251    }

252    /// <summary>
253    /// <para>
254    /// Determines whether this instance first is to the left of second.
255    /// </para>
256    /// <para></para>
257    /// </summary>
258    /// <param name="first">
259    /// <para>The first.</para>
260    /// <para></para>
261    /// </param>
262    /// <param name="second">
263    /// <para>The second.</para>
264    /// <para></para>
265    /// </param>
266    /// <returns>
267    /// <para>The bool</para>
268    /// <para></para>
269    /// </returns>
270    [MethodImpl(MethodImplOptions.AggressiveInlining)]
271    protected override bool FirstIsToLeftOfSecond(TLink first, TLink second)
272    {
273        ref var firstLink = ref GetLinkDataPartReference(first);
274        ref var secondLink = ref GetLinkDataPartReference(second);

```



```

278         return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
279             ↪ secondLink.Source, secondLink.Target);
280     }
281     /// <summary>
282     /// <para>
283     /// Determines whether this instance first is to the right of second.
284     /// </para>
285     /// <para></para>
286     /// </summary>
287     /// <param name="first">
288     /// <para>The first.</para>
289     /// <para></para>
290     /// </param>
291     /// <param name="second">
292     /// <para>The second.</para>
293     /// <para></para>
294     /// </param>
295     /// <returns>
296     /// <para>The bool</para>
297     /// <para></para>
298     /// </returns>
299     [MethodImpl(MethodImplOptions.AggressiveInlining)]
300     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
301     {
302         ref var firstLink = ref GetLinkDataPartReference(first);
303         ref var secondLink = ref GetLinkDataPartReference(second);
304         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
305             ↪ secondLink.Source, secondLink.Target);
306     }
307     /// <summary>
308     /// <para>
309     /// The zero.
310     /// </para>
311     /// <para></para>
312     /// </summary>
313     public TLink this[TLink index]
314     {
315         [MethodImpl(MethodImplOptions.AggressiveInlining)]
316         get
317         {
318             var root = GetTreeRoot();
319             if (GreaterOrEqualThan(index, GetSize(root)))
320             {
321                 return Zero;
322             }
323             while (!EqualToZero(root))
324             {
325                 var left = GetLeftOrDefault(root);
326                 var leftSize = GetSizeOrZero(left);
327                 if (LessThan(index, leftSize))
328                 {
329                     root = left;
330                     continue;
331                 }
332                 if (AreEqual(index, leftSize))
333                 {
334                     return root;
335                 }
336                 root = GetRightOrDefault(root);
337                 index = Subtract(index, Increment(leftSize));
338             }
339             return Zero; // TODO: Impossible situation exception (only if tree structure
340                 ↪ broken)
341         }
342     }
343     /// <summary>
344     /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
345     ↪ (концом).
346     /// </summary>
347     /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
348     /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
349     /// <returns>Индекс искомой связи.</returns>
350     [MethodImpl(MethodImplOptions.AggressiveInlining)]
351     public TLink Search(TLink source, TLink target)
352     {

```

```

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

```

```

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

```

```

504         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
505         {
506             return @break;
507         }
508     }
509     return @continue;
510 }
511
512 /// <summary>
513 /// <para>
514 /// Prints the node value using the specified node.
515 /// </para>
516 /// <para></para>
517 /// </summary>
518 /// <param name="node">
519 /// <para>The node.</para>
520 /// <para></para>
521 /// </param>
522 /// <param name="sb">
523 /// <para>The sb.</para>
524 /// <para></para>
525 /// </param>
526 [MethodImpl(MethodImplOptions.AggressiveInlining)]
527 protected override void PrintNodeValue(TLink node, StringBuilder sb)
528 {
529     ref var link = ref GetLinkDataPartReference(node);
530     sb.Append(' ');
531     sb.Append(link.Source);
532     sb.Append('-');
533     sb.Append('>');
534     sb.Append(link.Target);
535 }
536 }
537 }

```

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         /// <summary>
25         /// <para>
26         /// The default.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
31             ↳ UncheckedConverter<TLink, long>.Default;
32
33         /// <summary>
34         /// <para>
35         /// The break.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         protected readonly TLink Break;
40
41         /// <summary>
42         /// <para>
43         /// The continue.
44         /// </para>
45         /// </summary>
46         protected readonly TLink Continue;
47     }
48 }

```

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

```

418         {
419             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
420             root = GetRightOrDefault(root);
421         }
422     }
423     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
424 }
425
426 /// <summary>
427 /// <para>
428 /// Eaches the usage using the specified base.
429 /// </para>
430 /// <para></para>
431 /// </summary>
432 /// <param name="@base">
433 /// <para>The base.</para>
434 /// <para></para>
435 /// </param>
436 /// <param name="handler">
437 /// <para>The handler.</para>
438 /// <para></para>
439 /// </param>
440 /// <returns>
441 /// <para>The link</para>
442 /// <para></para>
443 /// </returns>
444 [MethodImpl(MethodImplOptions.AggressiveInlining)]
445 public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
446     ↳ EachUsageCore(@base, GetTreeRoot(), handler);
447
448 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
449 ↳ low-level MSIL stack.
450 /// <summary>
451 /// <para>
452 /// Eaches the usage core using the specified base.
453 /// </para>
454 /// <para></para>
455 /// </summary>
456 /// <param name="@base">
457 /// <para>The base.</para>
458 /// <para></para>
459 /// </param>
460 /// <param name="link">
461 /// <para>The link.</para>
462 /// <para></para>
463 /// </param>
464 /// <param name="handler">
465 /// <para>The handler.</para>
466 /// <para></para>
467 /// </param>
468 /// <returns>
469 /// <para>The continue.</para>
470 /// <para></para>
471 /// </returns>
472 [MethodImpl(MethodImplOptions.AggressiveInlining)]
473 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
474 {
475     var @continue = Continue;
476     if (EqualToZero(link))
477     {
478         return @continue;
479     }
480     var linkBasePart = GetBasePartValue(link);
481     var @break = Break;
482     if (GreaterThan(linkBasePart, @base))
483     {
484         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
485         {
486             return @break;
487         }
488     }
489     else if (LessThan(linkBasePart, @base))
490     {
491         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
492         {
493             return @break;
494         }
495     }
496 }

```

```

494     else //if (linkBasePart == @base)
495     {
496         if (AreEqual(handler(GetLinkValues(link)), @break))
497         {
498             return @break;
499         }
500         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
501         {
502             return @break;
503         }
504         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
505         {
506             return @break;
507         }
508     }
509     return @continue;
510 }
511
512 /// <summary>
513 /// <para>
514 /// Prints the node value using the specified node.
515 /// </para>
516 /// <para></para>
517 /// </summary>
518 /// <param name="node">
519 /// <para>The node.</para>
520 /// <para></para>
521 /// </param>
522 /// <param name="sb">
523 /// <para>The sb.</para>
524 /// <para></para>
525 /// </param>
526 [MethodImpl(MethodImplOptions.AggressiveInlining)]
527 protected override void PrintNodeValue(TLink node, StringBuilder sb)
528 {
529     ref var link = ref GetLinkDataPartReference(node);
530     sb.Append(' ');
531     sb.Append(link.Source);
532     sb.Append(' ');
533     sb.Append(' ');
534     sb.Append(link.Target);
535 }
536 }
537 }

```

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">

```

```

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

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

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

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

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

```

```

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

```

```

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

```

```

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 }

```

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

```

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

```

```

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

```



```

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

```

```

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

```

```

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

40
41        /// <summary>
42        /// <para>
43        /// Gets the left reference using the specified node.
44        /// </para>

```

```

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

```

```

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

```

```

193     /// </para>
194     /// <para></para>
195     /// </summary>
196     /// <param name="link">
197     /// <para>The link.</para>
198     /// <para></para>
199     /// </param>
200     /// <returns>
201     /// <para>The link</para>
202     /// <para></para>
203     /// </returns>
204     [MethodImpl(MethodImplOptions.AggressiveInlining)]
205     protected override TLink GetBasePartValue(TLink link) =>
206         ↪ GetLinkDataPartReference(link).Target;
207
208     /// <summary>
209     /// <para>
210     /// Determines whether this instance first is to the left of second.
211     /// </para>
212     /// <para></para>
213     /// </summary>
214     /// <param name="firstSource">
215     /// <para>The first source.</para>
216     /// <para></para>
217     /// </param>
218     /// <param name="firstTarget">
219     /// <para>The first target.</para>
220     /// <para></para>
221     /// </param>
222     /// <param name="secondSource">
223     /// <para>The second source.</para>
224     /// <para></para>
225     /// </param>
226     /// <param name="secondTarget">
227     /// <para>The second target.</para>
228     /// <para></para>
229     /// </param>
230     /// <returns>
231     /// <para>The bool</para>
232     /// <para></para>
233     /// </returns>
234     [MethodImpl(MethodImplOptions.AggressiveInlining)]
235     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236         ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
237         ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
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(firstTarget, secondTarget) ||
268         ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));

```

```

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>

```

```

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

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

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

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

108
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;

```



```

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

```

```

201    /// <para>The link</para>
202    /// <para></para>
203    /// </returns>
204    [MethodImpl(MethodImplOptions.AggressiveInlining)]
205    protected override TLink GetBasePartValue(TLink link) =>
206        ↪ GetLinkDataPartReference(link).Target;
207
208    /// <summary>
209    /// <para>
210    /// Determines whether this instance first is to the left of second.
211    /// </para>
212    /// <para></para>
213    /// </summary>
214    /// <param name="firstSource">
215    /// <para>The first source.</para>
216    /// <para></para>
217    /// </param>
218    /// <param name="firstTarget">
219    /// <para>The first target.</para>
220    /// <para></para>
221    /// </param>
222    /// <param name="secondSource">
223    /// <para>The second source.</para>
224    /// <para></para>
225    /// </param>
226    /// <param name="secondTarget">
227    /// <para>The second target.</para>
228    /// <para></para>
229    /// </param>
230    /// <returns>
231    /// <para>The bool</para>
232    /// <para></para>
233    /// </returns>
234    [MethodImpl(MethodImplOptions.AggressiveInlining)]
235    protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
236        ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
237        ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
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(firstTarget, secondTarget) ||
268        ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, 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>

```

```

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         /// <summary>
25         /// <para>
26         /// The default.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
31         ↪ UncheckedConverter<TLink, long>.Default;
32
33         /// <summary>
34         /// <para>
35         /// The break.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         protected readonly TLink Break;
40         /// <summary>
41         /// <para>
42         /// The continue.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         protected readonly TLink Continue;
47         /// <summary>
48         /// <para>
49         /// The links data parts.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         protected readonly byte* LinksDataParts;
54         /// <summary>
55         /// <para>
56         /// The links index parts.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         protected readonly byte* LinksIndexParts;
61         /// <summary>
62         /// <para>
63         /// The header.
64         /// </para>
65         /// <para></para>
66         /// </summary>
67         protected readonly byte* Header;
68     }
69 }

```

```

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

```

```

140    /// <para>The link.</para>
141    /// <para></para>
142    /// </param>
143    /// <returns>
144    /// <para>The link.</para>
145    /// <para></para>
146    /// </returns>
147    [MethodImpl(MethodImplOptions.AggressiveInlining)]
148    protected abstract TLink GetKeyPartValue(TLink link);
149
150    /// <summary>
151    /// <para>
152    /// Gets the link data part reference using the specified link.
153    /// </para>
154    /// <para></para>
155    /// </summary>
156    /// <param name="link">
157    /// <para>The link.</para>
158    /// <para></para>
159    /// </param>
160    /// <returns>
161    /// <para>A ref raw link data part of t link</para>
162    /// <para></para>
163    /// </returns>
164    [MethodImpl(MethodImplOptions.AggressiveInlining)]
165    protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↪ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
    ↪ _addressToInt64Converter.Convert(link)));
166
167    /// <summary>
168    /// <para>
169    /// Gets the link index part reference using the specified link.
170    /// </para>
171    /// <para></para>
172    /// </summary>
173    /// <param name="link">
174    /// <para>The link.</para>
175    /// <para></para>
176    /// </param>
177    /// <returns>
178    /// <para>A ref raw link index part of t link</para>
179    /// <para></para>
180    /// </returns>
181    [MethodImpl(MethodImplOptions.AggressiveInlining)]
182    protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
    ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
183
184    /// <summary>
185    /// <para>
186    /// Determines whether this instance first is to the left of second.
187    /// </para>
188    /// <para></para>
189    /// </summary>
190    /// <param name="first">
191    /// <para>The first.</para>
192    /// <para></para>
193    /// </param>
194    /// <param name="second">
195    /// <para>The second.</para>
196    /// <para></para>
197    /// </param>
198    /// <returns>
199    /// <para>The bool</para>
200    /// <para></para>
201    /// </returns>
202    [MethodImpl(MethodImplOptions.AggressiveInlining)]
203    protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
    ↪ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
204
205    /// <summary>
206    /// <para>
207    /// Determines whether this instance first is to the right of second.
208    /// </para>
209    /// <para></para>
210    /// </summary>
211    /// <param name="first">
212    /// <para>The first.</para>

```

```

213     /// <para></para>
214     /// </param>
215     /// <param name="second">
216     /// <para>The second.</para>
217     /// <para></para>
218     /// </param>
219     /// <returns>
220     /// <para>The bool</para>
221     /// <para></para>
222     /// </returns>
223     [MethodImpl(MethodImplOptions.AggressiveInlining)]
224     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
        ↪ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));

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

```

```

288     /// <returns>Индекс искомой связи.</returns>
289     [MethodImpl(MethodImplOptions.AggressiveInlining)]
290     public abstract TLink Search(TLink source, TLink target);
291
292     /// <summary>
293     /// <para>
294     /// Searches the core using the specified root.
295     /// </para>
296     /// <para></para>
297     /// </summary>
298     /// <param name="root">
299     /// <para>The root.</para>
300     /// <para></para>
301     /// </param>
302     /// <param name="key">
303     /// <para>The key.</para>
304     /// <para></para>
305     /// </param>
306     /// <returns>
307     /// <para>The zero.</para>
308     /// <para></para>
309     /// </returns>
310     [MethodImpl(MethodImplOptions.AggressiveInlining)]
311     protected TLink SearchCore(TLink root, TLink key)
312     {
313         while (!EqualToZero(root))
314         {
315             var rootKey = GetKeyPartValue(root);
316             if (LessThan(key, rootKey)) // node.Key < root.Key
317             {
318                 root = GetLeftOrDefault(root);
319             }
320             else if (GreaterThan(key, rootKey)) // node.Key > root.Key
321             {
322                 root = GetRightOrDefault(root);
323             }
324             else // node.Key == root.Key
325             {
326                 return root;
327             }
328         }
329         return Zero;
330     }
331
332     // TODO: Return indices range instead of references count
333     /// <summary>
334     /// <para>
335     /// Counts the usages using the specified link.
336     /// </para>
337     /// <para></para>
338     /// </summary>
339     /// <param name="link">
340     /// <para>The link.</para>
341     /// <para></para>
342     /// </param>
343     /// <returns>
344     /// <para>The link</para>
345     /// <para></para>
346     /// </returns>
347     [MethodImpl(MethodImplOptions.AggressiveInlining)]
348     public TLink CountUsages(TLink link) => GetSizeOrZero(GetTreeRoot(link));
349
350     /// <summary>
351     /// <para>
352     /// Eaches the usage using the specified base.
353     /// </para>
354     /// <para></para>
355     /// </summary>
356     /// <param name="@base">
357     /// <para>The base.</para>
358     /// <para></para>
359     /// </param>
360     /// <param name="handler">
361     /// <para>The handler.</para>
362     /// <para></para>
363     /// </param>
364     /// <returns>
365     /// <para>The link</para>

```

```

366 /// <para></para>
367 /// </returns>
368 [MethodImpl(MethodImplOptions.AggressiveInlining)]
369 public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
    ↳ EachUsageCore(@base, GetTreeRoot(@base), handler);
370
371 // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
    ↳ low-level MSIL stack.
372 /// <summary>
373 /// <para>
374 /// Eaches the usage core using the specified base.
375 /// </para>
376 /// <para></para>
377 /// </summary>
378 /// <param name="@base">
379 /// <para>The base.</para>
380 /// <para></para>
381 /// </param>
382 /// <param name="link">
383 /// <para>The link.</para>
384 /// <para></para>
385 /// </param>
386 /// <param name="handler">
387 /// <para>The handler.</para>
388 /// <para></para>
389 /// </param>
390 /// <returns>
391 /// <para>The continue.</para>
392 /// <para></para>
393 /// </returns>
394 [MethodImpl(MethodImplOptions.AggressiveInlining)]
395 private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
396 {
397     var @continue = Continue;
398     if (EqualToZero(link))
399     {
400         return @continue;
401     }
402     var @break = Break;
403     if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
404     {
405         return @break;
406     }
407     if (AreEqual(handler(GetLinkValues(link)), @break))
408     {
409         return @break;
410     }
411     if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
412     {
413         return @break;
414     }
415     return @continue;
416 }
417
418 /// <summary>
419 /// <para>
420 /// Prints the node value using the specified node.
421 /// </para>
422 /// <para></para>
423 /// </summary>
424 /// <param name="node">
425 /// <para>The node.</para>
426 /// <para></para>
427 /// </param>
428 /// <param name="sb">
429 /// <para>The sb.</para>
430 /// <para></para>
431 /// </param>
432 [MethodImpl(MethodImplOptions.AggressiveInlining)]
433 protected override void PrintNodeValue(TLink node, StringBuilder sb)
434 {
435     ref var link = ref GetLinkDataPartReference(node);
436     sb.Append(' ');
437     sb.Append(link.Source);
438     sb.Append('-');
439     sb.Append('>');
440     sb.Append(link.Target);
441 }

```



```
442 }
443 }
```

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         /// <summary>
25         /// <para>
26         /// The default.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
31         ↪ UncheckedConverter<TLink, long>.Default;
32
33         /// <summary>
34         /// <para>
35         /// The break.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         protected readonly TLink Break;
40
41         /// <summary>
42         /// <para>
43         /// The continue.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         protected readonly TLink Continue;
48
49         /// <summary>
50         /// <para>
51         /// The links data parts.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         protected readonly byte* LinksDataParts;
56
57         /// <summary>
58         /// <para>
59         /// The links index parts.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         protected readonly byte* LinksIndexParts;
64
65         /// <summary>
66         /// <para>
67         /// The header.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         protected readonly byte* Header;
72
73         /// <summary>
74         /// <para>
75         /// Initializes a new <see cref="InternalLinksSizeBalancedTreeMethodsBase"/> instance.
76         /// </para>
77         /// <para></para>
78         /// </summary>
79         /// <param name="constants">
```

```

74    /// <para>A constants.</para>
75    /// <para></para>
76    /// </param>
77    /// <param name="linksDataParts">
78    /// <para>A links data parts.</para>
79    /// <para></para>
80    /// </param>
81    /// <param name="linksIndexParts">
82    /// <para>A links index parts.</para>
83    /// <para></para>
84    /// </param>
85    /// <param name="header">
86    /// <para>A header.</para>
87    /// <para></para>
88    /// </param>
89    [MethodImpl(MethodImplOptions.AggressiveInlining)]
90    protected InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
    ↪    byte* linksDataParts, byte* linksIndexParts, byte* header)
91    {
92        LinksDataParts = linksDataParts;
93        LinksIndexParts = linksIndexParts;
94        Header = header;
95        Break = constants.Break;
96        Continue = constants.Continue;
97    }
98
99    /// <summary>
100    /// <para>
101    /// Gets the tree root 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 GetTreeRoot(TLink link);
115
116    /// <summary>
117    /// <para>
118    /// Gets the base part value using the specified link.
119    /// </para>
120    /// <para></para>
121    /// </summary>
122    /// <param name="link">
123    /// <para>The link.</para>
124    /// <para></para>
125    /// </param>
126    /// <returns>
127    /// <para>The link</para>
128    /// <para></para>
129    /// </returns>
130    [MethodImpl(MethodImplOptions.AggressiveInlining)]
131    protected abstract TLink GetBasePartValue(TLink link);
132
133    /// <summary>
134    /// <para>
135    /// Gets the key part value using the specified link.
136    /// </para>
137    /// <para></para>
138    /// </summary>
139    /// <param name="link">
140    /// <para>The link.</para>
141    /// <para></para>
142    /// </param>
143    /// <returns>
144    /// <para>The link</para>
145    /// <para></para>
146    /// </returns>
147    [MethodImpl(MethodImplOptions.AggressiveInlining)]
148    protected abstract TLink GetKeyPartValue(TLink link);
149
150    /// <summary>

```

```

151     /// <para>
152     /// Gets the link data part reference using the specified link.
153     /// </para>
154     /// <para></para>
155     /// </summary>
156     /// <param name="link">
157     /// <para>The link.</para>
158     /// <para></para>
159     /// </param>
160     /// <returns>
161     /// <para>A ref raw link data part of t link</para>
162     /// <para></para>
163     /// </returns>
164     [MethodImpl(MethodImplOptions.AggressiveInlining)]
165     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↳ AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
        ↳ _addressToInt64Converter.Convert(link)));

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

183
184     /// <summary>
185     /// <para>
186     /// Determines whether this instance first is to the left of second.
187     /// </para>
188     /// <para></para>
189     /// </summary>
190     /// <param name="first">
191     /// <para>The first.</para>
192     /// <para></para>
193     /// </param>
194     /// <param name="second">
195     /// <para>The second.</para>
196     /// <para></para>
197     /// </param>
198     /// <returns>
199     /// <para>The bool</para>
200     /// <para></para>
201     /// </returns>
202     [MethodImpl(MethodImplOptions.AggressiveInlining)]
203     protected override bool FirstIsToLeftOfSecond(TLink first, TLink second) =>
        ↳ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));

204
205     /// <summary>
206     /// <para>
207     /// Determines whether this instance first is to the right of second.
208     /// </para>
209     /// <para></para>
210     /// </summary>
211     /// <param name="first">
212     /// <para>The first.</para>
213     /// <para></para>
214     /// </param>
215     /// <param name="second">
216     /// <para>The second.</para>
217     /// <para></para>
218     /// </param>
219     /// <returns>
220     /// <para>The bool</para>
221     /// <para></para>
222     /// </returns>
223     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

```

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

```

```

375     /// </para>
376     /// <para></para>
377     /// </summary>
378     /// <param name="@base">
379     /// <para>The base.</para>
380     /// <para></para>
381     /// </param>
382     /// <param name="link">
383     /// <para>The link.</para>
384     /// <para></para>
385     /// </param>
386     /// <param name="handler">
387     /// <para>The handler.</para>
388     /// <para></para>
389     /// </param>
390     /// <returns>
391     /// <para>The continue.</para>
392     /// <para></para>
393     /// </returns>
394     [MethodImpl(MethodImplOptions.AggressiveInlining)]
395     private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
396     {
397         var @continue = Continue;
398         if (EqualToZero(link))
399         {
400             return @continue;
401         }
402         var @break = Break;
403         if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
404         {
405             return @break;
406         }
407         if (AreEqual(handler(GetLinkValues(link)), @break))
408         {
409             return @break;
410         }
411         if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
412         {
413             return @break;
414         }
415         return @continue;
416     }
417
418     /// <summary>
419     /// <para>
420     /// Prints the node value using the specified node.
421     /// </para>
422     /// <para></para>
423     /// </summary>
424     /// <param name="node">
425     /// <para>The node.</para>
426     /// <para></para>
427     /// </param>
428     /// <param name="sb">
429     /// <para>The sb.</para>
430     /// <para></para>
431     /// </param>
432     [MethodImpl(MethodImplOptions.AggressiveInlining)]
433     protected override void PrintNodeValue(TLink node, StringBuilder sb)
434     {
435         ref var link = ref GetLinkDataPartReference(node);
436         sb.Append(' ');
437         sb.Append(link.Source);
438         sb.Append('-');
439         sb.Append('>');
440         sb.Append(link.Target);
441     }
442 }
443 }

```

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         /// <summary>
23         /// <para>
24         /// The default.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
29             ↪ UncheckedConverter<TLink, long>.Default;
30         /// <summary>
31         /// <para>
32         /// The links data parts.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         private readonly byte* _linksDataParts;
37         /// <summary>
38         /// <para>
39         /// The links index parts.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         private readonly byte* _linksIndexParts;
44         /// <summary>
45         /// <para>
46         /// The break.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         protected readonly TLink Break;
51         /// <summary>
52         /// <para>
53         /// The continue.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         protected readonly TLink Continue;
58         /// <summary>
59         /// <para>
60         /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
61         /// </para>
62         /// <para></para>
63         /// </summary>
64         /// <param name="constants">
65         /// <para>A constants.</para>
66         /// <para></para>
67         /// </param>
68         /// <param name="linksDataParts">
69         /// <para>A links data parts.</para>
70         /// <para></para>
71         /// </param>
72         /// <param name="linksIndexParts">
73         /// <para>A links index parts.</para>
74         /// <para></para>
75         /// </param>
76         [MethodImpl(MethodImplOptions.AggressiveInlining)]
77         public InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants, byte*
78             ↪ linksDataParts, byte* linksIndexParts)
79         {
80             _linksDataParts = linksDataParts;
81             _linksIndexParts = linksIndexParts;
82             Break = constants.Break;
83             Continue = constants.Continue;
84         }
85     }
86     /// <summary>

```

```

85     /// <para>
86     /// Gets the link data part reference using the specified link.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="link">
91     /// <para>The link.</para>
92     /// <para></para>
93     /// </param>
94     /// <returns>
95     /// <para>A ref raw link data part of t link</para>
96     /// <para></para>
97     /// </returns>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
    ↪ AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (RawLinkDataPart<TLink>.SizeInBytes
    ↪ * _addressToInt64Converter.Convert(link)));

100
101     /// <summary>
102     /// <para>
103     /// Gets the link index part reference using the specified link.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     /// <param name="link">
108     /// <para>The link.</para>
109     /// <para></para>
110     /// </param>
111     /// <returns>
112     /// <para>A ref raw link index part of t link</para>
113     /// <para></para>
114     /// </returns>
115     [MethodImpl(MethodImplOptions.AggressiveInlining)]
116     protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
    ↪ ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
    ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));

117
118     /// <summary>
119     /// <para>
120     /// Gets the first using the specified head.
121     /// </para>
122     /// <para></para>
123     /// </summary>
124     /// <param name="head">
125     /// <para>The head.</para>
126     /// <para></para>
127     /// </param>
128     /// <returns>
129     /// <para>The link</para>
130     /// <para></para>
131     /// </returns>
132     [MethodImpl(MethodImplOptions.AggressiveInlining)]
133     protected override TLink GetFirst(TLink head) =>
    ↪ GetLinkIndexPartReference(head).RootAsSource;

134
135     /// <summary>
136     /// <para>
137     /// Gets the last using the specified head.
138     /// </para>
139     /// <para></para>
140     /// </summary>
141     /// <param name="head">
142     /// <para>The head.</para>
143     /// <para></para>
144     /// </param>
145     /// <returns>
146     /// <para>The link</para>
147     /// <para></para>
148     /// </returns>
149     [MethodImpl(MethodImplOptions.AggressiveInlining)]
150     protected override TLink GetLast(TLink head)
151     {
152         var first = GetLinkIndexPartReference(head).RootAsSource;
153         if (EqualToZero(first))
154         {
155             return first;
156         }
157         else

```



```

158     {
159         return GetPrevious(first);
160     }
161 }
162
163 /// <summary>
164 /// <para>
165 /// Gets the previous 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 GetPrevious(TLink element) =>
179     ↪ GetLinkIndexPartReference(element).LeftAsSource;
180
181 /// <summary>
182 /// <para>
183 /// Gets the next using the specified element.
184 /// </para>
185 /// <para></para>
186 /// </summary>
187 /// <param name="element">
188 /// <para>The element.</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 GetNext(TLink element) =>
197     ↪ GetLinkIndexPartReference(element).RightAsSource;
198
199 /// <summary>
200 /// <para>
201 /// Gets the size 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 /// <returns>
210 /// <para>The link</para>
211 /// <para></para>
212 /// </returns>
213 [MethodImpl(MethodImplOptions.AggressiveInlining)]
214 protected override TLink GetSize(TLink head) =>
215     ↪ GetLinkIndexPartReference(head).SizeAsSource;
216
217 /// <summary>
218 /// <para>
219 /// Sets the first 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)]
232 protected override void SetFirst(TLink head, TLink element) =>
233     ↪ GetLinkIndexPartReference(head).RootAsSource = element;
234
235 /// <summary>

```

```

232    /// <para>
233    /// Sets the last using the specified head.
234    /// </para>
235    /// <para></para>
236    /// </summary>
237    /// <param name="head">
238    /// <para>The head.</para>
239    /// <para></para>
240    /// </param>
241    /// <param name="element">
242    /// <para>The element.</para>
243    /// <para></para>
244    /// </param>
245    [MethodImpl(MethodImplOptions.AggressiveInlining)]
246    protected override void SetLast(TLink head, TLink element)
247    {
248        //var first = GetLinkIndexPartReference(head).RootAsSource;
249        //if (EqualToZero(first))
250        //{
251            SetFirst(head, element);
252        //}
253        //else
254        //{
255            SetPrevious(first, element);
256        //}
257    }
258
259    /// <summary>
260    /// <para>
261    /// Sets the previous 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="previous">
270    /// <para>The previous.</para>
271    /// <para></para>
272    /// </param>
273    [MethodImpl(MethodImplOptions.AggressiveInlining)]
274    protected override void SetPrevious(TLink element, TLink previous) =>
275        ↪ GetLinkIndexPartReference(element).LeftAsSource = previous;
276
277    /// <summary>
278    /// <para>
279    /// Sets the next using the specified element.
280    /// </para>
281    /// <para></para>
282    /// </summary>
283    /// <param name="element">
284    /// <para>The element.</para>
285    /// <para></para>
286    /// </param>
287    /// <param name="next">
288    /// <para>The next.</para>
289    /// <para></para>
290    /// </param>
291    [MethodImpl(MethodImplOptions.AggressiveInlining)]
292    protected override void SetNext(TLink element, TLink next) =>
293        ↪ GetLinkIndexPartReference(element).RightAsSource = next;
294
295    /// <summary>
296    /// <para>
297    /// Sets the size 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    /// <param name="size">
306    /// <para>The size.</para>
307    /// <para></para>
308    /// </param>
309    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

308     protected override void SetSize(TLink head, TLink size) =>
309         ↪ GetLinkIndexPartReference(head).SizeAsSource = size;
310
311     /// <summary>
312     /// <para>
313     /// Counts the usages using the specified head.
314     /// </para>
315     /// </summary>
316     /// <param name="head">
317     /// <para>The head.</para>
318     /// </param>
319     /// </returns>
320     /// <para>The link</para>
321     /// </returns>
322     [MethodImpl(MethodImplOptions.AggressiveInlining)]
323     public TLink CountUsages(TLink head) => GetSize(head);
324
325     /// <summary>
326     /// <para>
327     /// Gets the link values using the specified link index.
328     /// </para>
329     /// </summary>
330     /// <param name="linkIndex">
331     /// <para>The link index.</para>
332     /// </param>
333     /// </returns>
334     /// <para>A list of t link</para>
335     /// </returns>
336     [MethodImpl(MethodImplOptions.AggressiveInlining)]
337     protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
338     {
339         ref var link = ref GetLinkDataPartReference(linkIndex);
340         return new Link<TLink>(linkIndex, link.Source, link.Target);
341     }
342
343     /// <summary>
344     /// <para>
345     /// Eaches the usage using the specified source.
346     /// </para>
347     /// </summary>
348     /// <param name="source">
349     /// <para>The source.</para>
350     /// </param>
351     /// <param name="handler">
352     /// <para>The handler.</para>
353     /// </param>
354     /// </returns>
355     /// <para>The continue.</para>
356     /// </returns>
357     [MethodImpl(MethodImplOptions.AggressiveInlining)]
358     public TLink EachUsage(TLink source, Func<IList<TLink>, TLink> handler)
359     {
360         var @continue = Continue;
361         var @break = Break;
362         var current = GetFirst(source);
363         var first = current;
364         while (!EqualToZero(current))
365         {
366             if (AreEqual(handler(GetLinkValues(current)), @break))
367             {
368                 return @break;
369             }
370             current = GetNext(current);
371             if (AreEqual(current, first))
372             {
373                 return @continue;
374             }
375         }
376     }
377
378     }
379
380     }
381
382     }
383
384     }

```

```

385         return @continue;
386     }
387 }
388 }

```

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

```

1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.Split.Generic
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the internal links sources recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14     public unsafe class InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink> :
15         ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↳ cref="InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="linksDataParts">
29         /// <para>A links data parts.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="linksIndexParts">
33         /// <para>A links index parts.</para>
34         /// <para></para>
35         /// </param>
36         /// <param name="header">
37         /// <para>A header.</para>
38         /// <para></para>
39         /// </param>
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         public InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
42             ↳ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
43             ↳ base(constants, linksDataParts, linksIndexParts, header) { }
44
45         /// <summary>
46         /// <para>
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).LeftAsSource;
62
63         /// <summary>
64         /// <para>
65         /// Gets the right reference using the specified node.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         /// <param name="node">
70         /// <para>The node.</para>
71         /// <para></para>
72         /// </param>
73         /// </summary>

```

```

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>
136    /// <param name="node">
137    /// <para>The node.</para>
138    /// <para></para>
139    /// </param>
140    /// <param name="right">
141    /// <para>The right.</para>
142    /// <para></para>
143    /// </param>
144    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```

```

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

```

1.39 ./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.

```

```

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

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

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

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

```



```

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

```

```

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

```

```

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) =>
        ↪ 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>
        ↪ constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
        ↪ base(constants, linksDataParts, linksIndexParts, header) { }
41
42         /// <summary>
43         /// <para>

```

```

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

```

```

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

```

```

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

```

```

265         public override TLink Search(TLink source, TLink target) =>
266             ↪ SearchCore(GetTreeRoot(target), source);
267     }
}

```

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

```

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

```

```

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

```



```

142     /// <summary>
143     /// <para>
144     /// Gets the size using the specified node.
145     /// </para>
146     /// <para></para>
147     /// </summary>
148     /// <param name="node">
149     /// <para>The node.</para>
150     /// <para></para>
151     /// </param>
152     /// <returns>
153     /// <para>The link</para>
154     /// <para></para>
155     /// </returns>
156     [MethodImpl(MethodImplOptions.AggressiveInlining)]
157     protected override TLink GetSize(TLink node) =>
158         ↪ GetLinkIndexPartReference(node).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) =>
176         ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;
177
178     /// <summary>
179     /// <para>
180     /// Gets the tree root using the specified link.
181     /// </para>
182     /// <para></para>
183     /// </summary>
184     /// <param name="link">
185     /// <para>The link.</para>
186     /// <para></para>
187     /// </param>
188     /// <returns>
189     /// <para>The link</para>
190     /// <para></para>
191     /// </returns>
192     [MethodImpl(MethodImplOptions.AggressiveInlining)]
193     protected override TLink GetTreeRoot(TLink link) =>
194         ↪ GetLinkIndexPartReference(link).RootAsTarget;
195
196     /// <summary>
197     /// <para>
198     /// Gets the base part value using the specified link.
199     /// </para>
200     /// <para></para>
201     /// </summary>
202     /// <param name="link">
203     /// <para>The link.</para>
204     /// <para></para>
205     /// </param>
206     /// <returns>
207     /// <para>The link</para>
208     /// <para></para>
209     /// </returns>
210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
211     protected override TLink GetBasePartValue(TLink link) =>
212         ↪ GetLinkDataPartReference(link).Target;
213
214     /// <summary>
215     /// <para>
216     /// Gets the key part value using the specified link.
217     /// </para>
218     /// <para></para>

```

```

216     /// </summary>
217     /// <param name="link">
218     /// <para>The link.</para>
219     /// <para></para>
220     /// </param>
221     /// <returns>
222     /// <para>The link</para>
223     /// <para></para>
224     /// </returns>
225     [MethodImpl(MethodImplOptions.AggressiveInlining)]
226     protected override TLink GetKeyPartValue(TLink link) =>
227         ↪ GetLinkDataPartReference(link).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 GetLinkIndexPartReference(node);
243         link.LeftAsTarget = Zero;
244         link.RightAsTarget = Zero;
245         link.SizeAsTarget = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
267         ↪ SearchCore(GetTreeRoot(target), source);
268 }

```

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         /// <summary>
21         /// <para>
22         /// The create internal source tree methods.
23         /// </para>

```

```

24     /// <para></para>
25     /// </summary>
26     private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
27     /// <summary>
28     /// <para>
29     /// The create external source tree methods.
30     /// </para>
31     /// <para></para>
32     /// </summary>
33     private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
34     /// <summary>
35     /// <para>
36     /// The create internal target tree methods.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
41     /// <summary>
42     /// <para>
43     /// The create external target tree methods.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
48     /// <summary>
49     /// <para>
50     /// The header.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     private byte* _header;
55     /// <summary>
56     /// <para>
57     /// The links data parts.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     private byte* _linksDataParts;
62     /// <summary>
63     /// <para>
64     /// The links index parts.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     private byte* _linksIndexParts;
69
70     /// <summary>
71     /// <para>
72     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     /// <param name="dataMemory">
77     /// <para>A data memory.</para>
78     /// <para></para>
79     /// </param>
80     /// <param name="indexMemory">
81     /// <para>A index memory.</para>
82     /// <para></para>
83     /// </param>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     public SplitMemoryLinks(string dataMemory, string indexMemory) : this(new
86     ↪ FileMappedResizableDirectMemory(dataMemory), new
87     ↪ FileMappedResizableDirectMemory(indexMemory)) { }
88
89     /// <summary>
90     /// <para>
91     /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="dataMemory">
96     /// <para>A data memory.</para>
97     /// <para></para>
98     /// </param>
99     /// <param name="indexMemory">
100    /// <para>A index memory.</para>
101    /// <para></para>
102    /// </param>

```

```

100 /// </param>
101 [MethodImpl(MethodImplOptions.AggressiveInlining)]
102 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
103
104 /// <summary>
105 /// <para>
106 /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
107 /// </para>
108 /// <para></para>
109 /// </summary>
110 /// <param name="dataMemory">
111 /// <para>A data memory.</para>
112 /// <para></para>
113 /// </param>
114 /// <param name="indexMemory">
115 /// <para>A index memory.</para>
116 /// <para></para>
117 /// </param>
118 /// <param name="memoryReservationStep">
119 /// <para>A memory reservation step.</para>
120 /// <para></para>
121 /// </param>
122 [MethodImpl(MethodImplOptions.AggressiveInlining)]
123 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
    ↳ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
    ↳ IndexTreeType.Default, useLinkedList: true) { }
124
125 /// <summary>
126 /// <para>
127 /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
128 /// </para>
129 /// <para></para>
130 /// </summary>
131 /// <param name="dataMemory">
132 /// <para>A data memory.</para>
133 /// <para></para>
134 /// </param>
135 /// <param name="indexMemory">
136 /// <para>A index memory.</para>
137 /// <para></para>
138 /// </param>
139 /// <param name="memoryReservationStep">
140 /// <para>A memory reservation step.</para>
141 /// <para></para>
142 /// </param>
143 /// <param name="constants">
144 /// <para>A constants.</para>
145 /// <para></para>
146 /// </param>
147 [MethodImpl(MethodImplOptions.AggressiveInlining)]
148 public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
    ↳ this(dataMemory, indexMemory, memoryReservationStep, constants,
    ↳ IndexTreeType.Default, useLinkedList: true) { }
149
150 /// <summary>
151 /// <para>
152 /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
153 /// </para>
154 /// <para></para>
155 /// </summary>
156 /// <param name="dataMemory">
157 /// <para>A data memory.</para>
158 /// <para></para>
159 /// </param>
160 /// <param name="indexMemory">
161 /// <para>A index memory.</para>
162 /// <para></para>
163 /// </param>
164 /// <param name="memoryReservationStep">
165 /// <para>A memory reservation step.</para>
166 /// <para></para>
167 /// </param>
168 /// <param name="constants">
169 /// <para>A constants.</para>

```

```

170    /// <para></para>
171    /// </param>
172    /// <param name="indexTreeType">
173    /// <para>A index tree type.</para>
174    /// <para></para>
175    /// </param>
176    /// <param name="useLinkedList">
177    /// <para>A use linked list.</para>
178    /// <para></para>
179    /// </param>
180    [MethodImpl(MethodImplOptions.AggressiveInlining)]
181    public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
        ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↪ memoryReservationStep, constants, useLinkedList)
182    {
183        if (indexTreeType == IndexTreeType.SizeBalancedTree)
184        {
185            _createInternalSourceTreeMethods = () => new
        ↪ InternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
186            _createExternalSourceTreeMethods = () => new
        ↪ ExternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
187            _createInternalTargetTreeMethods = () => new
        ↪ InternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
188            _createExternalTargetTreeMethods = () => new
        ↪ ExternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
189        }
190        else
191        {
192            _createInternalSourceTreeMethods = () => new
        ↪ InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
193            _createExternalSourceTreeMethods = () => new
        ↪ ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
194            _createInternalTargetTreeMethods = () => new
        ↪ InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
195            _createExternalTargetTreeMethods = () => new
        ↪ ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
        ↪ _linksDataParts, _linksIndexParts, _header);
196        }
197        Init(dataMemory, indexMemory);
198    }
199
200    /// <summary>
201    /// <para>
202    /// Sets the pointers using the specified data memory.
203    /// </para>
204    /// <para></para>
205    /// </summary>
206    /// <param name="dataMemory">
207    /// <para>The data memory.</para>
208    /// <para></para>
209    /// </param>
210    /// <param name="indexMemory">
211    /// <para>The index memory.</para>
212    /// <para></para>
213    /// </param>
214    [MethodImpl(MethodImplOptions.AggressiveInlining)]
215    protected override void SetPointers(IResizableDirectMemory dataMemory,
        ↪ IResizableDirectMemory indexMemory)
216    {
217        _linksDataParts = (byte*)dataMemory.Pointer;
218        _linksIndexParts = (byte*)indexMemory.Pointer;
219        _header = _linksIndexParts;
220        if (_useLinkedList)
221        {
222            InternalSourcesListMethods = new
        ↪ InternalLinksSourcesLinkedListMethods<TLink>(Constants, _linksDataParts,
        ↪ _linksIndexParts);
223        }
224        else

```

```

225     {
226         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
227     }
228     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
229     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
230     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
231     UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_linksDataParts, _header);
232 }
233
234 /// <summary>
235 /// <para>
236 /// Resets the pointers.
237 /// </para>
238 /// <para></para>
239 /// </summary>
240 [MethodImpl(MethodImplOptions.AggressiveInlining)]
241 protected override void ResetPointers()
242 {
243     base.ResetPointers();
244     _linksDataParts = null;
245     _linksIndexParts = null;
246     _header = null;
247 }
248
249 /// <summary>
250 /// <para>
251 /// Gets the header reference.
252 /// </para>
253 /// <para></para>
254 /// </summary>
255 /// <returns>
256 /// <para>A ref links header of t link</para>
257 /// <para></para>
258 /// </returns>
259 [MethodImpl(MethodImplOptions.AggressiveInlining)]
260 protected override ref LinksHeader<TLink> GetHeaderReference() => ref
261     ↳ AsRef<LinksHeader<TLink>>(_header);
262
263 /// <summary>
264 /// <para>
265 /// Gets the link data part reference using the specified link index.
266 /// </para>
267 /// <para></para>
268 /// </summary>
269 /// <param name="linkIndex">
270 /// <para>The link index.</para>
271 /// <para></para>
272 /// </param>
273 /// <returns>
274 /// <para>A ref raw link data part of t link</para>
275 /// <para></para>
276 /// </returns>
277 [MethodImpl(MethodImplOptions.AggressiveInlining)]
278 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
279     ↳ => ref AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (LinkDataPartSizeInBytes *
280     ↳ ConvertToInt64(linkIndex)));
281
282 /// <summary>
283 /// <para>
284 /// Gets the link index part reference using the specified link index.
285 /// </para>
286 /// <para></para>
287 /// </summary>
288 /// <param name="linkIndex">
289 /// <para>The link index.</para>
290 /// <para></para>
291 /// </param>
292 /// <returns>
293 /// <para>A ref raw link index part of t link</para>
294 /// <para></para>
295 /// </returns>
296 [MethodImpl(MethodImplOptions.AggressiveInlining)]
297 protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
298     ↳ linkIndex) => ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
299     ↳ (LinkIndexPartSizeInBytes * ConvertToInt64(linkIndex)));
300 }

```

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         /// <summary>
26         /// <para>
27         /// The default.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         private static readonly EqualityComparer<TLink> _equalityComparer =
32             ↪ EqualityComparer<TLink>.Default;
33         /// <summary>
34         /// <para>
35         /// The default.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
40         /// <summary>
41         /// <para>
42         /// The default.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
47             ↪ UncheckedConverter<TLink, long>.Default;
48         /// <summary>
49         /// <para>
50         /// The default.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
55             ↪ UncheckedConverter<long, TLink>.Default;
56
57         /// <summary>
58         /// <para>
59         /// The zero.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         private static readonly TLink _zero = default;
64         /// <summary>
65         /// <para>
66         /// The zero.
67         /// </para>
68         /// <para></para>
69         /// </summary>
70         private static readonly TLink _one = Arithmetic.Increment(_zero);
71
72         /// <summary>Возвращает размер одной связи в байтах.</summary>
73         /// <remarks>
74         /// Используется только во вне класса, не рекомендуется использовать внутри.
75         /// Так как во вне не обязательно будет доступен unsafe C#.
76         /// </remarks>
77         public static readonly long LinkDataPartSizeInBytes = RawLinkDataPart<TLink>.SizeInBytes;

```

```

76     /// <summary>
77     /// <para>
78     /// The size in bytes.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     public static readonly long LinkIndexPartSizeInBytes =
83         ↳ RawLinkIndexPart<TLink>.SizeInBytes;
84
85     /// <summary>
86     /// <para>
87     /// The size in bytes.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
92
93     /// <summary>
94     /// <para>
95     /// The default links size step.
96     /// </para>
97     /// <para></para>
98     /// </summary>
99     public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
100
101     /// <summary>
102     /// <para>
103     /// The data memory.
104     /// </para>
105     /// <para></para>
106     /// </summary>
107     protected readonly IResizableDirectMemory _dataMemory;
108
109     /// <summary>
110     /// <para>
111     /// The index memory.
112     /// </para>
113     /// <para></para>
114     /// </summary>
115     protected readonly IResizableDirectMemory _indexMemory;
116
117     /// <summary>
118     /// <para>
119     /// The use linked list.
120     /// </para>
121     /// <para></para>
122     /// </summary>
123     protected readonly bool _useLinkedList;
124
125     /// <summary>
126     /// <para>
127     /// The data memory reservation step in bytes.
128     /// </para>
129     /// <para></para>
130     /// </summary>
131     protected readonly long _dataMemoryReservationStepInBytes;
132
133     /// <summary>
134     /// <para>
135     /// The index memory reservation step in bytes.
136     /// </para>
137     /// <para></para>
138     /// </summary>
139     protected readonly long _indexMemoryReservationStepInBytes;
140
141     /// <summary>
142     /// <para>
143     /// The internal sources list methods.
144     /// </para>
145     /// <para></para>
146     /// </summary>
147     protected InternalLinksSourcesLinkedListMethods<TLink> InternalSourcesListMethods;
148
149     /// <summary>
150     /// <para>
151     /// The internal sources tree methods.
152     /// </para>
153     /// <para></para>
154     /// </summary>
155     protected ILinksTreeMethods<TLink> InternalSourcesTreeMethods;

```



```

154     /// <para></para>
155     /// </summary>
156     protected ILinksTreeMethods<TLink> ExternalSourcesTreeMethods;
157     /// <summary>
158     /// <para>
159     /// The internal targets tree methods.
160     /// </para>
161     /// <para></para>
162     /// </summary>
163     protected ILinksTreeMethods<TLink> InternalTargetsTreeMethods;
164     /// <summary>
165     /// <para>
166     /// The external targets tree methods.
167     /// </para>
168     /// <para></para>
169     /// </summary>
170     protected ILinksTreeMethods<TLink> ExternalTargetsTreeMethods;
171     // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
172     // → нужно использовать не список а дерево, так как так можно быстрее проверить на
173     // → наличие связи внутри
174     /// <summary>
175     /// <para>
176     /// The unused links list methods.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     protected ILinksListMethods<TLink> UnusedLinksListMethods;
181     /// <summary>
182     /// Возвращает общее число связей находящихся в хранилище.
183     /// </summary>
184     protected virtual TLink Total
185     {
186         [MethodImpl(MethodImplOptions.AggressiveInlining)]
187         get
188         {
189             ref var header = ref GetHeaderReference();
190             return Subtract(header.AllocatedLinks, header.FreeLinks);
191         }
192     }
193     /// <summary>
194     /// <para>
195     /// Gets the constants value.
196     /// </para>
197     /// <para></para>
198     /// </summary>
199     public virtual LinksConstants<TLink> Constants
200     {
201         [MethodImpl(MethodImplOptions.AggressiveInlining)]
202         get;
203     }
204
205     /// <summary>
206     /// <para>
207     /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
208     /// </para>
209     /// <para></para>
210     /// </summary>
211     /// <param name="dataMemory">
212     /// <para>A data memory.</para>
213     /// <para></para>
214     /// </param>
215     /// <param name="indexMemory">
216     /// <para>A index memory.</para>
217     /// <para></para>
218     /// </param>
219     /// <param name="memoryReservationStep">
220     /// <para>A memory reservation step.</para>
221     /// <para></para>
222     /// </param>
223     /// <param name="constants">
224     /// <para>A constants.</para>
225     /// <para></para>
226     /// </param>
227     /// <param name="useLinkedList">
228     /// <para>A use linked list.</para>
229     /// <para></para>
230     /// </param>

```

```

231 [MethodImpl(MethodImplOptions.AggressiveInlining)]
232 protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants, bool
    ↳ useLinkedList)
233 {
234     _dataMemory = dataMemory;
235     _indexMemory = indexMemory;
236     _dataMemoryReservationStepInBytes = memoryReservationStep * LinkDataPartSizeInBytes;
237     _indexMemoryReservationStepInBytes = memoryReservationStep *
        ↳ LinkIndexPartSizeInBytes;
238     useLinkedList = useLinkedList;
239     Constants = constants;
240 }
241
242 /// <summary>
243 /// <para>
244 /// Initializes a new <see cref="SplitMemoryLinksBase"/> instance.
245 /// </para>
246 /// <para></para>
247 /// </summary>
248 /// <param name="dataMemory">
249 /// <para>A data memory.</para>
250 /// <para></para>
251 /// </param>
252 /// <param name="indexMemory">
253 /// <para>A index memory.</para>
254 /// <para></para>
255 /// </param>
256 /// <param name="memoryReservationStep">
257 /// <para>A memory reservation step.</para>
258 /// <para></para>
259 /// </param>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 protected SplitMemoryLinksBase(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
    ↳ memoryReservationStep, Default<LinksConstants<TLink>>.Instance, useLinkedList: true)
    ↳ { }
262
263 /// <summary>
264 /// <para>
265 /// Inits the data memory.
266 /// </para>
267 /// <para></para>
268 /// </summary>
269 /// <param name="dataMemory">
270 /// <para>The data memory.</para>
271 /// <para></para>
272 /// </param>
273 /// <param name="indexMemory">
274 /// <para>The index memory.</para>
275 /// <para></para>
276 /// </param>
277 [MethodImpl(MethodImplOptions.AggressiveInlining)]
278 protected virtual void Init(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory)
279 {
280     // Read allocated links from header
281     if (indexMemory.ReservedCapacity < LinkHeaderSizeInBytes)
282     {
283         indexMemory.ReservedCapacity = LinkHeaderSizeInBytes;
284     }
285     SetPointers(dataMemory, indexMemory);
286     ref var header = ref GetHeaderReference();
287     var allocatedLinks = ConvertToInt64(header.AllocatedLinks);
288     // Adjust reserved capacity
289     var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
290     if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
291     {
292         minimumDataReservedCapacity = dataMemory.UsedCapacity;
293     }
294     if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
295     {
296         minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
297     }
298     var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
299     if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
300     {
301         minimumIndexReservedCapacity = indexMemory.UsedCapacity;

```

```

302     }
303     if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
304     {
305         minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
306     }
307     // Check for alignment
308     if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
309     {
310         minimumDataReservedCapacity = ((minimumDataReservedCapacity /
311             ↪ _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
312             ↪ _dataMemoryReservationStepInBytes;
313     }
314     if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)
315     {
316         minimumIndexReservedCapacity = ((minimumIndexReservedCapacity /
317             ↪ _indexMemoryReservationStepInBytes) * _indexMemoryReservationStepInBytes) +
318             ↪ _indexMemoryReservationStepInBytes;
319     }
320     if (dataMemory.ReservedCapacity != minimumDataReservedCapacity)
321     {
322         dataMemory.ReservedCapacity = minimumDataReservedCapacity;
323     }
324     if (indexMemory.ReservedCapacity != minimumIndexReservedCapacity)
325     {
326         indexMemory.ReservedCapacity = minimumIndexReservedCapacity;
327     }
328     SetPointers(dataMemory, indexMemory);
329     header = ref GetHeaderReference();
330     // Ensure correctness _memory.UsedCapacity over _header->AllocatedLinks
331     // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
332     dataMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
333         ↪ LinkDataPartSizeInBytes) + LinkDataPartSizeInBytes; // First link is read only
334         ↪ zero link.
335     indexMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
336         ↪ LinkIndexPartSizeInBytes) + LinkHeaderSizeInBytes;
337     // Ensure correctness _memory.ReservedLinks over _header->ReservedCapacity
338     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
339     header.ReservedLinks = ConvertToAddress((dataMemory.ReservedCapacity -
340         ↪ LinkDataPartSizeInBytes) / LinkDataPartSizeInBytes);
341 }
342
343 /// <summary>
344 /// <para>
345 /// Counts the restrictions.
346 /// </para>
347 /// <para></para>
348 /// </summary>
349 /// <param name="restrictions">
350 /// <para>The restrictions.</para>
351 /// <para></para>
352 /// </param>
353 /// <exception cref="NotSupportedException">
354 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
355 /// <para></para>
356 /// </exception>
357 /// <returns>
358 /// <para>The link</para>
359 /// <para></para>
360 /// </returns>
361 [MethodImpl(MethodImplOptions.AggressiveInlining)]
362 public virtual TLink Count(ICollection<TLink> restrictions)
363 {
364     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
365     if (restrictions.Count == 0)
366     {
367         return Total;
368     }
369     var constants = Constants;
370     var any = constants.Any;
371     var index = restrictions[constants.IndexPart];
372     if (restrictions.Count == 1)
373     {
374         if (AreEqual(index, any))
375         {
376             return Total;
377         }
378         return Exists(index) ? GetOne() : GetZero();
379     }
380 }

```

```

372 if (restrictions.Count == 2)
373 {
374     var value = restrictions[1];
375     if (AreEqual(index, any))
376     {
377         if (AreEqual(value, any))
378         {
379             return Total; // Any - как отсутствие ограничения
380         }
381         var externalReferencesRange = constants.ExternalReferencesRange;
382         if (externalReferencesRange.HasValue &&
383             ⇨ externalReferencesRange.Value.Contains(value))
384         {
385             return Add(ExternalSourcesTreeMethods.CountUsages(value),
386                 ⇨ ExternalTargetsTreeMethods.CountUsages(value));
387         }
388         else
389         {
390             if (_useLinkedList)
391             {
392                 return Add(InternalSourcesListMethods.CountUsages(value),
393                     ⇨ InternalTargetsTreeMethods.CountUsages(value));
394             }
395             else
396             {
397                 return Add(InternalSourcesTreeMethods.CountUsages(value),
398                     ⇨ InternalTargetsTreeMethods.CountUsages(value));
399             }
400         }
401     }
402     else
403     {
404         if (!Exists(index))
405         {
406             return GetZero();
407         }
408         if (AreEqual(value, any))
409         {
410             return GetOne();
411         }
412         ref var storedLinkValue = ref GetLinkDataPartReference(index);
413         if (AreEqual(storedLinkValue.Source, value) ||
414             ⇨ AreEqual(storedLinkValue.Target, value))
415         {
416             return GetOne();
417         }
418         return GetZero();
419     }
420 }
421 if (restrictions.Count == 3)
422 {
423     var externalReferencesRange = constants.ExternalReferencesRange;
424     var source = restrictions[constants.SourcePart];
425     var target = restrictions[constants.TargetPart];
426     if (AreEqual(index, any))
427     {
428         if (AreEqual(source, any) && AreEqual(target, any))
429         {
430             return Total;
431         }
432         else if (AreEqual(source, any))
433         {
434             if (externalReferencesRange.HasValue &&
435                 ⇨ externalReferencesRange.Value.Contains(target))
436             {
437                 return ExternalTargetsTreeMethods.CountUsages(target);
438             }
439             else
440             {
441                 return InternalTargetsTreeMethods.CountUsages(target);
442             }
443         }
444         else if (AreEqual(target, any))
445         {
446             if (externalReferencesRange.HasValue &&
447                 ⇨ externalReferencesRange.Value.Contains(source))
448             {

```

```

442         return ExternalSourcesTreeMethods.CountUsages(source);
443     }
444     else
445     {
446         if (_useLinkedList)
447         {
448             return InternalSourcesListMethods.CountUsages(source);
449         }
450         else
451         {
452             return InternalSourcesTreeMethods.CountUsages(source);
453         }
454     }
455 }
456 else //if(source != Any && target != Any)
457 {
458     // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
459     TLink link;
460     if (externalReferencesRange.HasValue)
461     {
462         if (externalReferencesRange.Value.Contains(source) &&
463             ↳ externalReferencesRange.Value.Contains(target))
464         {
465             link = ExternalSourcesTreeMethods.Search(source, target);
466         }
467         else if (externalReferencesRange.Value.Contains(source))
468         {
469             link = InternalTargetsTreeMethods.Search(source, target);
470         }
471         else if (externalReferencesRange.Value.Contains(target))
472         {
473             if (_useLinkedList)
474             {
475                 link = ExternalSourcesTreeMethods.Search(source, target);
476             }
477             else
478             {
479                 link = InternalSourcesTreeMethods.Search(source, target);
480             }
481         }
482         else
483         {
484             if (_useLinkedList ||
485                 ↳ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
486                 ↳ InternalTargetsTreeMethods.CountUsages(target)))
487             {
488                 link = InternalTargetsTreeMethods.Search(source, target);
489             }
490             else
491             {
492                 link = InternalSourcesTreeMethods.Search(source, target);
493             }
494         }
495     }
496     }
497     }
498     }
499     }
500     }
501     }
502     }
503     }
504     return AreEqual(link, constants.Null) ? GetZero() : GetOne();
505 }
506 }
507 else
508 {
509     if (!Exists(index))
510     {
511         return GetZero();
512     }
513     if (AreEqual(source, any) && AreEqual(target, any))
514     {

```

```

515         return GetOne();
516     }
517     ref var storedLinkValue = ref GetLinkDataPartReference(index);
518     if (!AreEqual(source, any) && !AreEqual(target, any))
519     {
520         if (AreEqual(storedLinkValue.Source, source) &&
521             ↪ AreEqual(storedLinkValue.Target, target))
522         {
523             return GetOne();
524         }
525         return GetZero();
526     }
527     var value = default(TLink);
528     if (AreEqual(source, any))
529     {
530         value = target;
531     }
532     if (AreEqual(target, any))
533     {
534         value = source;
535     }
536     if (AreEqual(storedLinkValue.Source, value) ||
537         ↪ AreEqual(storedLinkValue.Target, value))
538     {
539         return GetOne();
540     }
541     return GetZero();
542 }
543 }
544
545 /// <summary>
546 /// <para>
547 /// Eaches the handler.
548 /// </para>
549 /// <para></para>
550 /// </summary>
551 /// <param name="handler">
552 /// <para>The handler.</para>
553 /// <para></para>
554 /// </param>
555 /// <param name="restrictions">
556 /// <para>The restrictions.</para>
557 /// <para></para>
558 /// </param>
559 /// <exception cref="NotSupportedException">
560 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
561 /// <para></para>
562 /// </exception>
563 /// <returns>
564 /// <para>The link</para>
565 /// <para></para>
566 /// </returns>
567 [MethodImpl(MethodImplOptions.AggressiveInlining)]
568 public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
569 {
570     var constants = Constants;
571     var @break = constants.Break;
572     if (restrictions.Count == 0)
573     {
574         for (var link = GetOne(); LessOrEqualThan(link,
575             ↪ GetHeaderReference().AllocatedLinks); link = Increment(link))
576         {
577             if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
578             {
579                 return @break;
580             }
581         }
582         return @break;
583     }
584     var @continue = constants.Continue;
585     var any = constants.Any;
586     var index = restrictions[constants.IndexPart];
587     if (restrictions.Count == 1)
588     {
589         if (AreEqual(index, any))

```

```

589     {
590         return Each(handler, Array.Empty<TLink>());
591     }
592     if (!Exists(index))
593     {
594         return @continue;
595     }
596     return handler(GetLinkStruct(index));
597 }
598 if (restrictions.Count == 2)
599 {
600     var value = restrictions[1];
601     if (AreEqual(index, any))
602     {
603         if (AreEqual(value, any))
604         {
605             return Each(handler, Array.Empty<TLink>());
606         }
607         if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
608         {
609             return @break;
610         }
611         return Each(handler, new Link<TLink>(index, any, value));
612     }
613     else
614     {
615         if (!Exists(index))
616         {
617             return @continue;
618         }
619         if (AreEqual(value, any))
620         {
621             return handler(GetLinkStruct(index));
622         }
623         ref var storedLinkValue = ref GetLinkDataPartReference(index);
624         if (AreEqual(storedLinkValue.Source, value) ||
625             AreEqual(storedLinkValue.Target, value))
626         {
627             return handler(GetLinkStruct(index));
628         }
629         return @continue;
630     }
631 }
632 if (restrictions.Count == 3)
633 {
634     var externalReferencesRange = constants.ExternalReferencesRange;
635     var source = restrictions[constants.SourcePart];
636     var target = restrictions[constants.TargetPart];
637     if (AreEqual(index, any))
638     {
639         if (AreEqual(source, any) && AreEqual(target, any))
640         {
641             return Each(handler, Array.Empty<TLink>());
642         }
643         else if (AreEqual(source, any))
644         {
645             if (externalReferencesRange.HasValue &&
646                 externalReferencesRange.Value.Contains(target))
647             {
648                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
649             }
650             else
651             {
652                 return InternalTargetsTreeMethods.EachUsage(target, handler);
653             }
654         }
655         else if (AreEqual(target, any))
656         {
657             if (externalReferencesRange.HasValue &&
658                 externalReferencesRange.Value.Contains(source))
659             {
660                 return ExternalSourcesTreeMethods.EachUsage(source, handler);
661             }
662             else
663             {
664                 if (_useLinkedList)
665                 {
666                     return InternalSourcesListMethods.EachUsage(source, handler);
667                 }
668             }
669         }
670     }
671 }

```

```

665     }
666     else
667     {
668         return InternalSourcesTreeMethods.EachUsage(source, handler);
669     }
670 }
671 }
672 else //if(source != Any && target != Any)
673 {
674     TLink link;
675     if (externalReferencesRange.HasValue)
676     {
677         if (externalReferencesRange.Value.Contains(source) &&
678             ↪ externalReferencesRange.Value.Contains(target))
679         {
680             link = ExternalSourcesTreeMethods.Search(source, target);
681         }
682         else if (externalReferencesRange.Value.Contains(source))
683         {
684             link = InternalTargetsTreeMethods.Search(source, target);
685         }
686         else if (externalReferencesRange.Value.Contains(target))
687         {
688             if (_useLinkedList)
689             {
690                 link = ExternalSourcesTreeMethods.Search(source, target);
691             }
692             else
693             {
694                 link = InternalSourcesTreeMethods.Search(source, target);
695             }
696         }
697         else
698         {
699             if (_useLinkedList ||
700                 ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
701                 ↪ InternalTargetsTreeMethods.CountUsages(target)))
702             {
703                 link = InternalTargetsTreeMethods.Search(source, target);
704             }
705             else
706             {
707                 link = InternalSourcesTreeMethods.Search(source, target);
708             }
709         }
710     }
711     else
712     {
713         if (_useLinkedList ||
714             ↪ GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
715             ↪ InternalTargetsTreeMethods.CountUsages(target)))
716         {
717             link = InternalTargetsTreeMethods.Search(source, target);
718         }
719         else
720         {
721             link = InternalSourcesTreeMethods.Search(source, target);
722         }
723     }
724     return AreEqual(link, constants.Null) ? @continue :
725         ↪ handler(GetLinkStruct(link));
726 }
727 }
728 else
729 {
730     if (!Exists(index))
731     {
732         return @continue;
733     }
734     if (AreEqual(source, any) && AreEqual(target, any))
735     {
736         return handler(GetLinkStruct(index));
737     }
738     ref var storedLinkValue = ref GetLinkDataPartReference(index);
739     if (!AreEqual(source, any) && !AreEqual(target, any))
740     {
741         if (AreEqual(storedLinkValue.Source, source) &&
742             AreEqual(storedLinkValue.Target, target))

```



```

737         {
738             return handler(GetLinkStruct(index));
739         }
740         return @continue;
741     }
742     var value = default(TLink);
743     if (AreEqual(source, any))
744     {
745         value = target;
746     }
747     if (AreEqual(target, any))
748     {
749         value = source;
750     }
751     if (AreEqual(storedLinkValue.Source, value) ||
752         AreEqual(storedLinkValue.Target, value))
753     {
754         return handler(GetLinkStruct(index));
755     }
756     return @continue;
757 }
758 }
759 throw new NotSupportedException("Другие размеры и способы ограничений не
    ↳ поддерживаются.");
760 }
761
762 /// <remarks>
763 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
764 /// ↳ в другом месте (но не в менеджере памяти, а в логике Links)
765 /// </remarks>
766 [MethodImpl(MethodImplOptions.AggressiveInlining)]
767 public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
768 {
769     var constants = Constants;
770     var @null = constants.Null;
771     var externalReferencesRange = constants.ExternalReferencesRange;
772     var linkIndex = restrictions[constants.IndexPart];
773     ref var link = ref GetLinkDataPartReference(linkIndex);
774     var source = link.Source;
775     var target = link.Target;
776     ref var header = ref GetHeaderReference();
777     ref var rootAsSource = ref header.RootAsSource;
778     ref var rootAsTarget = ref header.RootAsTarget;
779     // Будет корректно работать только в том случае, если пространство выделенной связи
780     // ↳ предварительно заполнено нулями
781     if (!AreEqual(source, @null))
782     {
783         if (externalReferencesRange.HasValue &&
784             ↳ externalReferencesRange.Value.Contains(source))
785         {
786             ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
787         }
788         else
789         {
790             if (_useLinkedList)
791             {
792                 InternalSourcesListMethods.Detach(source, linkIndex);
793             }
794             else
795             {
796                 InternalSourcesTreeMethods.Detach(ref
797                     ↳ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
798             }
799         }
800     }
801     if (!AreEqual(target, @null))
802     {
803         if (externalReferencesRange.HasValue &&
804             ↳ externalReferencesRange.Value.Contains(target))
805         {
806             ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
807         }
808         else
809         {
810             InternalTargetsTreeMethods.Detach(ref
811                 ↳ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
812         }
813     }
814 }

```

```

808 source = link.Source = substitution[constants.SourcePart];
809 target = link.Target = substitution[constants.TargetPart];
810 if (!AreEqual(source, @null))
811 {
812     if (externalReferencesRange.HasValue &&
813         ↪ externalReferencesRange.Value.Contains(source))
814     {
815         ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
816     }
817     else
818     {
819         if (_useLinkedList)
820         {
821             InternalSourcesListMethods.AttachAsLast(source, linkIndex);
822         }
823         else
824         {
825             InternalSourcesTreeMethods.Attach(ref
826                 ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
827         }
828     }
829 }
830 if (!AreEqual(target, @null))
831 {
832     if (externalReferencesRange.HasValue &&
833         ↪ externalReferencesRange.Value.Contains(target))
834     {
835         ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
836     }
837     else
838     {
839         InternalTargetsTreeMethods.Attach(ref
840             ↪ GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
841     }
842 }
843 return linkIndex;
844 }
845
846 /// <remarks>
847 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
848 ↪ пространство
849 /// </remarks>
850 [MethodImpl(MethodImplOptions.AggressiveInlining)]
851 public virtual TLink Create(ICollection<TLink> restrictions)
852 {
853     ref var header = ref GetHeaderReference();
854     var freeLink = header.FirstFreeLink;
855     if (!AreEqual(freeLink, Constants.Null))
856     {
857         UnusedLinksListMethods.Detach(freeLink);
858     }
859     else
860     {
861         var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
862         if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
863         {
864             throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
865         }
866         if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
867         {
868             _dataMemory.ReservedCapacity += _dataMemory.ReservationStepInBytes;
869             _indexMemory.ReservedCapacity += _indexMemory.ReservationStepInBytes;
870             SetPointers(_dataMemory, _indexMemory);
871             header = ref GetHeaderReference();
872             header.ReservedLinks = ConvertToAddress(_dataMemory.ReservedCapacity /
873                 ↪ LinkDataPartSizeInBytes);
874         }
875         freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
876         _dataMemory.UsedCapacity += LinkDataPartSizeInBytes;
877         _indexMemory.UsedCapacity += LinkIndexPartSizeInBytes;
878     }
879     return freeLink;
880 }
881
882 /// <summary>
883 /// <para>
884 /// Deletes the restrictions.
885 /// </para>

```

```

880 /// <para></para>
881 /// </summary>
882 /// <param name="restrictions">
883 /// <para>The restrictions.</para>
884 /// <para></para>
885 /// </param>
886 [MethodImpl(MethodImplOptions.AggressiveInlining)]
887 public virtual void Delete(IList<TLink> restrictions)
888 {
889     ref var header = ref GetHeaderReference();
890     var link = restrictions[Constants.IndexPart];
891     if (LessThan(link, header.AllocatedLinks))
892     {
893         UnusedLinksListMethods.AttachAsFirst(link);
894     }
895     else if (AreEqual(link, header.AllocatedLinks))
896     {
897         header.AllocatedLinks = Decrement(header.AllocatedLinks);
898         _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
899         _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
900         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
901         // → пока не дойдём до первой существующей связи
902         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
903         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
904             → IsUnusedLink(header.AllocatedLinks))
905         {
906             UnusedLinksListMethods.Detach(header.AllocatedLinks);
907             header.AllocatedLinks = Decrement(header.AllocatedLinks);
908             _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
909             _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
910         }
911     }
912 }
913
914 /// <summary>
915 /// <para>
916 /// Gets the link struct using the specified link index.
917 /// </para>
918 /// <para></para>
919 /// </summary>
920 /// <param name="linkIndex">
921 /// <para>The link index.</para>
922 /// <para></para>
923 /// </param>
924 /// <returns>
925 /// <para>A list of t link</para>
926 /// <para></para>
927 /// </returns>
928 [MethodImpl(MethodImplOptions.AggressiveInlining)]
929 public IList<TLink> GetLinkStruct(TLink linkIndex)
930 {
931     ref var link = ref GetLinkDataPartReference(linkIndex);
932     return new Link<TLink>(linkIndex, link.Source, link.Target);
933 }
934
935 /// <remarks>
936 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
937 → адрес реально поменялся
938 ///
939 /// Указатель this.links может быть в том же месте,
940 /// так как 0-я связь не используется и имеет такой же размер как Header,
941 /// поэтому header размещается в том же месте, что и 0-я связь
942 /// </remarks>
943 [MethodImpl(MethodImplOptions.AggressiveInlining)]
944 protected abstract void SetPointers(IResizableDirectMemory dataMemory,
945     → IResizableDirectMemory indexMemory);
946
947 /// <summary>
948 /// <para>
949 /// Resets the pointers.
950 /// </para>
951 /// <para></para>
952 /// </summary>
953 [MethodImpl(MethodImplOptions.AggressiveInlining)]
954 protected virtual void ResetPointers()
955 {
956     InternalSourcesListMethods = null;
957     InternalSourcesTreeMethods = null;
958 }

```

```

954     ExternalSourcesTreeMethods = null;
955     InternalTargetsTreeMethods = null;
956     ExternalTargetsTreeMethods = null;
957     UnusedLinksListMethods = null;
958 }
959
960 /// <summary>
961 /// <para>
962 /// Gets the header reference.
963 /// </para>
964 /// <para></para>
965 /// </summary>
966 /// <returns>
967 /// <para>A ref links header of t link</para>
968 /// <para></para>
969 /// </returns>
970 [MethodImpl(MethodImplOptions.AggressiveInlining)]
971 protected abstract ref LinksHeader<TLink> GetHeaderReference();
972
973 /// <summary>
974 /// <para>
975 /// Gets the link data part reference using the specified link index.
976 /// </para>
977 /// <para></para>
978 /// </summary>
979 /// <param name="linkIndex">
980 /// <para>The link index.</para>
981 /// <para></para>
982 /// </param>
983 /// <returns>
984 /// <para>A ref raw link data part of t link</para>
985 /// <para></para>
986 /// </returns>
987 [MethodImpl(MethodImplOptions.AggressiveInlining)]
988 protected abstract ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex);
989
990 /// <summary>
991 /// <para>
992 /// Gets the link index part reference using the specified link index.
993 /// </para>
994 /// <para></para>
995 /// </summary>
996 /// <param name="linkIndex">
997 /// <para>The link index.</para>
998 /// <para></para>
999 /// </param>
1000 /// <returns>
1001 /// <para>A ref raw link index part of t link</para>
1002 /// <para></para>
1003 /// </returns>
1004 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1005 protected abstract ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
    ↪ linkIndex);
1006
1007 /// <summary>
1008 /// <para>
1009 /// Determines whether this instance exists.
1010 /// </para>
1011 /// <para></para>
1012 /// </summary>
1013 /// <param name="link">
1014 /// <para>The link.</para>
1015 /// <para></para>
1016 /// </param>
1017 /// <returns>
1018 /// <para>The bool</para>
1019 /// <para></para>
1020 /// </returns>
1021 [MethodImpl(MethodImplOptions.AggressiveInlining)]
1022 protected virtual bool Exists(TLink link)
1023     => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
1024     && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
1025     && !IsUnusedLink(link);
1026
1027 /// <summary>
1028 /// <para>
1029 /// Determines whether this instance is unused link.
1030 /// </para>

```

```

1031    /// <para></para>
1032    /// </summary>
1033    /// <param name="linkIndex">
1034    /// <para>The link index.</para>
1035    /// <para></para>
1036    /// </param>
1037    /// <returns>
1038    /// <para>The bool</para>
1039    /// <para></para>
1040    /// </returns>
1041    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1042    protected virtual bool IsUnusedLink(TLink linkIndex)
1043    {
1044        if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
1045        {
1046            ↪ is not needed
1047            // TODO: Reduce access to memory in different location (should be enough to use
1048            ↪ just linkIndexPart)
1049            ref var linkDataPart = ref GetLinkDataPartReference(linkIndex);
1050            ref var linkIndexPart = ref GetLinkIndexPartReference(linkIndex);
1051            return AreEqual(linkIndexPart.SizeAsTarget, default) &&
1052            ↪ !AreEqual(linkDataPart.Source, default);
1053        }
1054        else
1055        {
1056            return true;
1057        }
1058    }
1059    /// <summary>
1060    /// <para>
1061    /// Gets the one.
1062    /// </para>
1063    /// <para></para>
1064    /// </summary>
1065    /// <returns>
1066    /// <para>The link</para>
1067    /// <para></para>
1068    /// </returns>
1069    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1070    protected virtual TLink GetOne() => _one;
1071    /// <summary>
1072    /// <para>
1073    /// Gets the zero.
1074    /// </para>
1075    /// <para></para>
1076    /// </summary>
1077    /// <returns>
1078    /// <para>The link</para>
1079    /// <para></para>
1080    /// </returns>
1081    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1082    protected virtual TLink GetZero() => default;
1083    /// <summary>
1084    /// <para>
1085    /// Determines whether this instance are equal.
1086    /// </para>
1087    /// <para></para>
1088    /// </summary>
1089    /// <param name="first">
1090    /// <para>The first.</para>
1091    /// <para></para>
1092    /// </param>
1093    /// <param name="second">
1094    /// <para>The second.</para>
1095    /// <para></para>
1096    /// </param>
1097    /// <returns>
1098    /// <para>The bool</para>
1099    /// <para></para>
1100    /// </returns>
1101    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1102    protected virtual bool AreEqual(TLink first, TLink second) =>
1103    ↪ _equalityComparer.Equals(first, second);
1104    /// <summary>

```

```

1105     /// <para>
1106     /// Determines whether this instance less than.
1107     /// </para>
1108     /// <para></para>
1109     /// </summary>
1110     /// <param name="first">
1111     /// <para>The first.</para>
1112     /// <para></para>
1113     /// </param>
1114     /// <param name="second">
1115     /// <para>The second.</para>
1116     /// <para></para>
1117     /// </param>
1118     /// <returns>
1119     /// <para>The bool</para>
1120     /// <para></para>
1121     /// </returns>
1122     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1123     protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
1124     ↪ second) < 0;
1125
1126     /// <summary>
1127     /// <para>
1128     /// Determines whether this instance less or equal than.
1129     /// </para>
1130     /// <para></para>
1131     /// </summary>
1132     /// <param name="first">
1133     /// <para>The first.</para>
1134     /// <para></para>
1135     /// </param>
1136     /// <param name="second">
1137     /// <para>The second.</para>
1138     /// <para></para>
1139     /// </param>
1140     /// <returns>
1141     /// <para>The bool</para>
1142     /// <para></para>
1143     /// </returns>
1144     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1145     protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
1146     ↪ _comparer.Compare(first, second) <= 0;
1147
1148     /// <summary>
1149     /// <para>
1150     /// Determines whether this instance greater than.
1151     /// </para>
1152     /// <para></para>
1153     /// </summary>
1154     /// <param name="first">
1155     /// <para>The first.</para>
1156     /// <para></para>
1157     /// </param>
1158     /// <param name="second">
1159     /// <para>The second.</para>
1160     /// <para></para>
1161     /// </param>
1162     /// <returns>
1163     /// <para>The bool</para>
1164     /// <para></para>
1165     /// </returns>
1166     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1167     protected virtual bool GreaterThan(TLink first, TLink second) =>
1168     ↪ _comparer.Compare(first, second) > 0;
1169
1170     /// <summary>
1171     /// <para>
1172     /// Determines whether this instance greater or equal than.
1173     /// </para>
1174     /// <para></para>
1175     /// </summary>
1176     /// <param name="first">
1177     /// <para>The first.</para>
1178     /// <para></para>
1179     /// </param>
1180     /// <param name="second">
1181     /// <para>The second.</para>
1182     /// <para></para>
1183     /// </param>
1184     /// <returns>
1185     /// <para>The bool</para>
1186     /// <para></para>
1187     /// </returns>
1188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1189     protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
1190     ↪ _comparer.Compare(first, second) >= 0;

```

```

1180    /// </param>
1181    /// <returns>
1182    /// <para>The bool</para>
1183    /// <para></para>
1184    /// </returns>
1185    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1186    protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
1187        ↪ _comparer.Compare(first, second) >= 0;
1188
1189    /// <summary>
1190    /// <para>
1191    /// Converts the to int 64 using the specified value.
1192    /// </para>
1193    /// <para></para>
1194    /// </summary>
1195    /// <param name="value">
1196    /// <para>The value.</para>
1197    /// <para></para>
1198    /// </param>
1199    /// <returns>
1200    /// <para>The long</para>
1201    /// <para></para>
1202    /// </returns>
1203    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1204    protected virtual long ConvertToInt64(TLink value) =>
1205        ↪ _addressToInt64Converter.Convert(value);
1206
1207    /// <summary>
1208    /// <para>
1209    /// Converts the to address using the specified value.
1210    /// </para>
1211    /// <para></para>
1212    /// </summary>
1213    /// <param name="value">
1214    /// <para>The value.</para>
1215    /// <para></para>
1216    /// </param>
1217    /// <returns>
1218    /// <para>The link</para>
1219    /// <para></para>
1220    /// </returns>
1221    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1222    protected virtual TLink ConvertToAddress(long value) =>
1223        ↪ _int64ToAddressConverter.Convert(value);
1224
1225    /// <summary>
1226    /// <para>
1227    /// Adds the first.
1228    /// </para>
1229    /// <para></para>
1230    /// </summary>
1231    /// <param name="first">
1232    /// <para>The first.</para>
1233    /// <para></para>
1234    /// </param>
1235    /// <param name="second">
1236    /// <para>The second.</para>
1237    /// <para></para>
1238    /// </param>
1239    /// <returns>
1240    /// <para>The link</para>
1241    /// <para></para>
1242    /// </returns>
1243    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1244    protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
1245        ↪ second);
1246
1247    /// <summary>
1248    /// <para>
1249    /// Subtracts the first.
1250    /// </para>
1251    /// <para></para>
1252    /// </summary>
1253    /// <param name="first">
1254    /// <para>The first.</para>
1255    /// <para></para>
1256    /// </param>
1257    /// <param name="second">

```

```

1254    /// <para>The second.</para>
1255    /// <para></para>
1256    /// </param>
1257    /// <returns>
1258    /// <para>The link</para>
1259    /// <para></para>
1260    /// </returns>
1261    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1262    protected virtual TLink Subtract(TLink first, TLink second) =>
        ↪ Arithmetic<TLink>.Subtract(first, second);

1263
1264    /// <summary>
1265    /// <para>
1266    /// Increments the link.
1267    /// </para>
1268    /// <para></para>
1269    /// </summary>
1270    /// <param name="link">
1271    /// <para>The link.</para>
1272    /// <para></para>
1273    /// </param>
1274    /// <returns>
1275    /// <para>The link</para>
1276    /// <para></para>
1277    /// </returns>
1278    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1279    protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
1280
1281    /// <summary>
1282    /// <para>
1283    /// Decrements the link.
1284    /// </para>
1285    /// <para></para>
1286    /// </summary>
1287    /// <param name="link">
1288    /// <para>The link.</para>
1289    /// <para></para>
1290    /// </param>
1291    /// <returns>
1292    /// <para>The link</para>
1293    /// <para></para>
1294    /// </returns>
1295    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1296    protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
1297
1298    #region Disposable
1299
1300    /// <summary>
1301    /// <para>
1302    /// Gets the allow multiple dispose calls value.
1303    /// </para>
1304    /// <para></para>
1305    /// </summary>
1306    protected override bool AllowMultipleDisposeCalls
1307    {
1308        [MethodImpl(MethodImplOptions.AggressiveInlining)]
1309        get => true;
1310    }
1311
1312    /// <summary>
1313    /// <para>
1314    /// Disposes the manual.
1315    /// </para>
1316    /// <para></para>
1317    /// </summary>
1318    /// <param name="manual">
1319    /// <para>The manual.</para>
1320    /// <para></para>
1321    /// </param>
1322    /// <param name="wasDisposed">
1323    /// <para>The was disposed.</para>
1324    /// <para></para>
1325    /// </param>
1326    [MethodImpl(MethodImplOptions.AggressiveInlining)]
1327    protected override void Dispose(bool manual, bool wasDisposed)
1328    {
1329        if (!wasDisposed)
1330        {

```



```

1331         ResetPointers();
1332         _dataMemory.DisposeIfPossible();
1333         _indexMemory.DisposeIfPossible();
1334     }
1335 }
1336
1337 #endregion
1338 }
1339 }

```

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         /// <summary>
22         /// <para>
23         /// The default.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
28             ↳ UncheckedConverter<TLink, long>.Default;
29
30         /// <summary>
31         /// <para>
32         /// The links.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         private readonly byte* _links;
37
38         /// <summary>
39         /// <para>
40         /// The header.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         private readonly byte* _header;
45
46         /// <summary>
47         /// <para>
48         /// Initializes a new <see cref="UnusedLinksListMethods"> instance.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <param name="links">
53         /// <para>A links.</para>
54         /// <para></para>
55         /// </param>
56         /// <param name="header">
57         /// <para>A header.</para>
58         /// <para></para>
59         /// </param>
60         [MethodImpl(MethodImplOptions.AggressiveInlining)]
61         public UnusedLinksListMethods(byte* links, byte* header)
62         {
63             _links = links;
64             _header = header;
65         }
66
67         /// <summary>
68         /// <para>
69         /// Gets the header reference.

```

```

67     /// </para>
68     /// <para></para>
69     /// </summary>
70     /// <returns>
71     /// <para>A ref links header of t link</para>
72     /// <para></para>
73     /// </returns>
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
76     ↪ AsRef<LinksHeader<TLink>>(_header);
77
78     /// <summary>
79     /// <para>
80     /// Gets the link data part reference using the specified link.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     /// <param name="link">
85     /// <para>The link.</para>
86     /// <para></para>
87     /// </param>
88     /// <returns>
89     /// <para>A ref raw link data part of t link</para>
90     /// <para></para>
91     /// </returns>
92     [MethodImpl(MethodImplOptions.AggressiveInlining)]
93     protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
94     ↪ AsRef<RawLinkDataPart<TLink>>(_links + (RawLinkDataPart<TLink>.SizeInBytes *
95     ↪ _addressToInt64Converter.Convert(link)));
96
97     /// <summary>
98     /// <para>
99     /// Gets the first.
100    /// </para>
101    /// <para></para>
102    /// </summary>
103    /// <returns>
104    /// <para>The link</para>
105    /// <para></para>
106    /// </returns>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
109
110    /// <summary>
111    /// <para>
112    /// Gets the last.
113    /// </para>
114    /// <para></para>
115    /// </summary>
116    /// <returns>
117    /// <para>The link</para>
118    /// <para></para>
119    /// </returns>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
122
123    /// <summary>
124    /// <para>
125    /// Gets the previous using the specified element.
126    /// </para>
127    /// <para></para>
128    /// </summary>
129    /// <param name="element">
130    /// <para>The element.</para>
131    /// <para></para>
132    /// </param>
133    /// <returns>
134    /// <para>The link</para>
135    /// <para></para>
136    /// </returns>
137    [MethodImpl(MethodImplOptions.AggressiveInlining)]
138    protected override TLink GetPrevious(TLink element) =>
139    ↪ GetLinkDataPartReference(element).Source;
140
141    /// <summary>
142    /// <para>
143    /// Gets the next using the specified element.
144    /// </para>

```

```

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

```

```

215     /// </summary>
216     /// <param name="element">
217     /// <para>The element.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="next">
221     /// <para>The next.</para>
222     /// <para></para>
223     /// </param>
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     protected override void SetNext(TLink element, TLink next) =>
226         ↪ GetLinkDataPartReference(element).Target = next;
227
228     /// <summary>
229     /// <para>
230     /// Sets the size using the specified size.
231     /// </para>
232     /// <para></para>
233     /// </summary>
234     /// <param name="size">
235     /// <para>The size.</para>
236     /// <para></para>
237     /// </param>
238     [MethodImpl(MethodImplOptions.AggressiveInlining)]
239     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
240 }

```

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         /// <summary>
19         /// <para>
20         /// The default.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private static readonly EqualityComparer<TLink> _equalityComparer =
25             ↪ EqualityComparer<TLink>.Default;
26
27         /// <summary>
28         /// <para>
29         /// The size.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         public static readonly long SizeInBytes = Structure<RawLinkDataPart<TLink>>.Size;
34
35         /// <summary>
36         /// <para>
37         /// The source.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         public TLink Source;
42
43         /// <summary>
44         /// <para>
45         /// The target.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         public TLink Target;
50
51         /// <summary>

```

```

50     /// <para>
51     /// Determines whether this instance equals.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     /// <param name="obj">
56     /// <para>The obj.</para>
57     /// <para></para>
58     /// </param>
59     /// <returns>
60     /// <para>The bool</para>
61     /// <para></para>
62     /// </returns>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     public override bool Equals(object obj) => obj is RawLinkDataPart<TLink> link ?
        ↪ Equals(link) : false;
65
66     /// <summary>
67     /// <para>
68     /// Determines whether this instance equals.
69     /// </para>
70     /// <para></para>
71     /// </summary>
72     /// <param name="other">
73     /// <para>The other.</para>
74     /// <para></para>
75     /// </param>
76     /// <returns>
77     /// <para>The bool</para>
78     /// <para></para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public bool Equals(RawLinkDataPart<TLink> other)
82         => _equalityComparer.Equals(Source, other.Source)
83         && _equalityComparer.Equals(Target, other.Target);
84
85     /// <summary>
86     /// <para>
87     /// Gets the hash code.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <returns>
92     /// <para>The int</para>
93     /// <para></para>
94     /// </returns>
95     [MethodImpl(MethodImplOptions.AggressiveInlining)]
96     public override int GetHashCode() => (Source, Target).GetHashCode();
97
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public static bool operator ==(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
        ↪ right) => left.Equals(right);
100
101     [MethodImpl(MethodImplOptions.AggressiveInlining)]
102     public static bool operator !=(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
        ↪ right) => !(left == right);
103 }
104 }

```

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         /// <summary>

```

```

19     /// <para>
20     /// The default.
21     /// </para>
22     /// <para></para>
23     /// </summary>
24     private static readonly EqualityComparer<TLink> _equalityComparer =
25         ↳ EqualityComparer<TLink>.Default;
26
27     /// <summary>
28     /// <para>
29     /// The size.
30     /// </para>
31     /// <para></para>
32     /// </summary>
33     public static readonly long SizeInBytes = Structure<RawLinkIndexPart<TLink>>.Size;
34
35     /// <summary>
36     /// <para>
37     /// The root as source.
38     /// </para>
39     /// <para></para>
40     /// </summary>
41     public TLink RootAsSource;
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 root as target.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     public TLink RootAsTarget;
70     /// <summary>
71     /// <para>
72     /// The left as target.
73     /// </para>
74     /// <para></para>
75     /// </summary>
76     public TLink LeftAsTarget;
77     /// <summary>
78     /// <para>
79     /// The right as target.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     public TLink RightAsTarget;
84     /// <summary>
85     /// <para>
86     /// The size as target.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     public TLink SizeAsTarget;
91
92     /// <summary>
93     /// <para>
94     /// Determines whether this instance equals.
95     /// </para>
96     /// <para></para>
97     /// </summary>

```

```

97     /// <param name="obj">
98     /// <para>The obj.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The bool</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    public override bool Equals(object obj) => obj is RawLinkIndexPart<TLink> link ?
        ↳ Equals(link) : false;

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

145
146    [MethodImpl(MethodImplOptions.AggressiveInlining)]
147    public static bool operator ==(RawLinkIndexPart<TLink> left, RawLinkIndexPart<TLink>
        ↳ right) => left.Equals(right);

148
149    [MethodImpl(MethodImplOptions.AggressiveInlining)]
150    public static bool operator !=(RawLinkIndexPart<TLink> left, RawLinkIndexPart<TLink>
        ↳ right) => !(left == right);

151 }
152 }

```

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)
    ↳ : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65     {
66         LinksDataParts = linksDataParts;
67         LinksIndexParts = linksIndexParts;
68         Header = header;
69     }
70
71     /// <summary>
72     /// <para>
73     /// Gets the zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <returns>
78     /// <para>The link</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override TLink GetZero() => 0U;
83
84     /// <summary>
85     /// <para>
86     /// Determines whether this instance equal to zero.
87     /// </para>
88     /// <para></para>
89     /// </summary>

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

1.48 ./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>

```

```

20    /// <para>
21    /// The links data parts.
22    /// </para>
23    /// <para></para>
24    /// </summary>
25    protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26    /// <summary>
27    /// <para>
28    /// The links index parts.
29    /// </para>
30    /// <para></para>
31    /// </summary>
32    protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33    /// <summary>
34    /// <para>
35    /// The header.
36    /// </para>
37    /// <para></para>
38    /// </summary>
39    protected new readonly LinksHeader<TLink>* Header;
40
41    /// <summary>
42    /// <para>
43    /// Initializes a new <see cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
44    ↪ instance.
45    /// </para>
46    /// <para></para>
47    /// </summary>
48    /// <param name="constants">
49    /// <para>A constants.</para>
50    /// <para></para>
51    /// </param>
52    /// <param name="linksDataParts">
53    /// <para>A links data parts.</para>
54    /// <para></para>
55    /// </param>
56    /// <param name="linksIndexParts">
57    /// <para>A links index parts.</para>
58    /// <para></para>
59    /// </param>
60    /// <param name="header">
61    /// <para>A header.</para>
62    /// <para></para>
63    /// </param>
64    [MethodImpl(MethodImplOptions.AggressiveInlining)]
65    protected UInt32ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
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>

```

```

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

```

```

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

```



```

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

```

```

326     /// </returns>
327     [MethodImpl(MethodImplOptions.AggressiveInlining)]
328     protected override TLink Add(TLink first, TLink second) => first + second;
329
330     /// <summary>
331     /// <para>
332     /// Subtracts the first.
333     /// </para>
334     /// <para></para>
335     /// </summary>
336     /// <param name="first">
337     /// <para>The first.</para>
338     /// <para></para>
339     /// </param>
340     /// <param name="second">
341     /// <para>The second.</para>
342     /// <para></para>
343     /// </param>
344     /// <returns>
345     /// <para>The link</para>
346     /// <para></para>
347     /// </returns>
348     [MethodImpl(MethodImplOptions.AggressiveInlining)]
349     protected override TLink Subtract(TLink first, TLink second) => first - second;
350
351     /// <summary>
352     /// <para>
353     /// Gets the header reference.
354     /// </para>
355     /// <para></para>
356     /// </summary>
357     /// <returns>
358     /// <para>A ref links header of t link</para>
359     /// <para></para>
360     /// </returns>
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;
363
364     /// <summary>
365     /// <para>
366     /// Gets the link data part reference using the specified link.
367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="link">
371     /// <para>The link.</para>
372     /// <para></para>
373     /// </param>
374     /// <returns>
375     /// <para>A ref raw link data part of t link</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
380         ↪ ref LinksDataParts[link];
381
382     /// <summary>
383     /// <para>
384     /// Gets the link index part reference using the specified link.
385     /// </para>
386     /// <para></para>
387     /// </summary>
388     /// <param name="link">
389     /// <para>The link.</para>
390     /// <para></para>
391     /// </param>
392     /// <returns>
393     /// <para>A ref raw link index part of t link</para>
394     /// <para></para>
395     /// </returns>
396     [MethodImpl(MethodImplOptions.AggressiveInlining)]
397     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
398         ↪ ref LinksIndexParts[link];
399
400     /// <summary>
401     /// <para>
402     /// Determines whether this instance first is to the left of second.
403     /// </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,
422             ↪ secondLink.Source, secondLink.Target);
423     }
424     /// <summary>
425     /// <para>
426     /// Determines whether this instance first is to the right of second.
427     /// </para>
428     /// <para></para>
429     /// </summary>
430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The bool</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
444     {
445         ref var firstLink = ref LinksDataParts[first];
446         ref var secondLink = ref LinksDataParts[second];
447         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
448             ↪ secondLink.Source, secondLink.Target);
449     }
450 }

```

1.49 ./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 :
16         ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">

```

```

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

```

```

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">

```

```

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

```

```

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

```

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

```

1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links sources size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksSourcesSizeBalancedTreeMethods :
16         ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="UInt32ExternalLinksSourcesSizeBalancedTreeMethods"/>
21         /// instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">

```

```

32    /// <para>A links index parts.</para>
33    /// <para></para>
34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public UInt32 ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
    ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↪ linksIndexParts, header) { }

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

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

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

```



```

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

```

```

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

```

```

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,
    ↪ TLink secondSource, TLink secondTarget)
266     => firstSource > secondSource || firstSource == secondSource && firstTarget >
    ↪ secondTarget;

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

```

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

```

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

```

```

public
    ↳ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↳ linksIndexParts, header) { }

    /// <summary>
    /// <para>
    /// Gets the left reference using the specified node.
    /// </para>
    /// </summary>
    /// <param name="node">
    /// <para>The node.</para>
    /// </param>
    /// <returns>
    /// <para>The ref link</para>
    /// </returns>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override ref TLink GetLeftReference(TLink node) => ref
        ↳ LinksIndexParts[node].LeftAsTarget;

    /// <summary>
    /// <para>
    /// Gets the right reference using the specified node.
    /// </para>
    /// </summary>
    /// <param name="node">
    /// <para>The node.</para>
    /// </param>
    /// <returns>
    /// <para>The ref link</para>
    /// </returns>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override ref TLink GetRightReference(TLink node) => ref
        ↳ LinksIndexParts[node].RightAsTarget;

    /// <summary>
    /// <para>
    /// Gets the left using the specified node.
    /// </para>
    /// </summary>
    /// <param name="node">
    /// <para>The node.</para>
    /// </param>
    /// <returns>
    /// <para>The link</para>
    /// </returns>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;

    /// <summary>
    /// <para>
    /// Gets the right using the specified node.
    /// </para>
    /// </summary>
    /// <param name="node">
    /// <para>The node.</para>
    /// </param>
    /// <returns>
    /// <para>The link</para>
    /// </returns>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;

    /// <summary>
    /// <para>

```

```

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

```

```

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

```

```

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

```

1.52 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods

```

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

```

```

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

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

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

```



```

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

```

```

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

```

```

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

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Data.Doublets.Memory.Split.Generic;
3 using TLink = System.UInt32;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Memory.Split.Specific
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the int 32 internal links recursionless size balanced tree methods base.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
16    public unsafe abstract class UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
17    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
18    {
19        /// <summary>
20        /// <para>
21        /// The links data parts.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26        /// <summary>
27        /// <para>
28        /// The links index parts.
29        /// </para>
30        /// <para></para>
31        /// </summary>
32        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33        /// <summary>
34        /// <para>
35        /// The header.
36        /// </para>
37        /// <para></para>
38        /// </summary>
39        protected new readonly LinksHeader<TLink>* Header;
40        /// <summary>
41        /// <para>
42        /// Initializes a new <see
43        ↪ cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
44        /// </para>
45        /// <para></para>
46        /// </summary>
47        /// <param name="constants">
48        /// <para>A constants.</para>
49        /// <para></para>
50        /// </param>
51        /// <param name="linksDataParts">
52        /// <para>A links data parts.</para>
53        /// <para></para>
54        /// </param>
55        /// <param name="linksIndexParts">
56        /// <para>A links index parts.</para>
57        /// <para></para>
58        /// </param>
59        /// <param name="header">

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

```

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

```

```

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

```

```

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

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

```

```

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

```

```

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

```

1.55 ./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         /// <summary>
19         /// <para>
20         /// The links data parts.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private readonly RawLinkDataPart<TLink>* _linksDataParts;
25
26         /// <summary>
27         /// <para>
28         /// The links index parts.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
33
34         /// <summary>
35         /// <para>
36         /// Initializes a new <see cref="UInt32InternalLinksSourcesLinkedListMethods"> instance.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         /// <param name="constants">

```

```

39     /// <para>A constants.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="linksDataParts">
43     /// <para>A links data parts.</para>
44     /// <para></para>
45     /// </param>
46     /// <param name="linksIndexParts">
47     /// <para>A links index parts.</para>
48     /// <para></para>
49     /// </param>
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
52     ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
53     {
54         _linksDataParts = linksDataParts;
55         _linksIndexParts = linksIndexParts;
56     }
57
58     /// <summary>
59     /// <para>
60     /// Gets the link data part reference using the specified link.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <param name="link">
65     /// <para>The link.</para>
66     /// <para></para>
67     /// </param>
68     /// <returns>
69     /// <para>A ref raw link data part of t link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
74     ↪ ref _linksDataParts[link];
75
76     /// <summary>
77     /// <para>
78     /// Gets the link index part reference using the specified link.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="link">
83     /// <para>The link.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>A ref raw link index part of t link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
92     ↪ ref _linksIndexParts[link];
93 }
94 }

```

1.56 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt32;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 32 internal links sources recursionless size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
16    ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>

```

```

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

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

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

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

```



```

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

```

```

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

```

```

243         link.LeftAsSource = Zero;
244         link.RightAsSource = Zero;
245         link.SizeAsSource = Zero;
246     }
247
248     /// <summary>
249     /// <para>
250     /// Searches the source.
251     /// </para>
252     /// <para></para>
253     /// </summary>
254     /// <param name="source">
255     /// <para>The source.</para>
256     /// <para></para>
257     /// </param>
258     /// <param name="target">
259     /// <para>The target.</para>
260     /// <para></para>
261     /// </param>
262     /// <returns>
263     /// <para>The link</para>
264     /// <para></para>
265     /// </returns>
266     public override TLink Search(TLink source, TLink target) =>
        ↪ SearchCore(GetTreeRoot(source), target);
267 }
268 }

```

1.57 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethod

```

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

```

```

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

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

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

```

```

122     /// <para></para>
123     /// </param>
124     [MethodImpl(MethodImplOptions.AggressiveInlining)]
125     protected override void SetLeft(TLink node, TLink left) =>
126         ↳ LinksIndexParts[node].LeftAsSource = left;
127
128     /// <summary>
129     /// <para>
130     /// Sets the right using the specified node.
131     /// </para>
132     /// <para></para>
133     /// </summary>
134     /// <param name="node">
135     /// <para>The node.</para>
136     /// </param>
137     /// <param name="right">
138     /// <para>The right.</para>
139     /// <para></para>
140     /// </param>
141     [MethodImpl(MethodImplOptions.AggressiveInlining)]
142     protected override void SetRight(TLink node, TLink right) =>
143         ↳ LinksIndexParts[node].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 using the specified node.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="node">
186     /// <para>The node.</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) =>
267         ↪ SearchCore(GetTreeRoot(source), target);
268 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt32;
3

```

```

4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 32 internal links targets recursionless size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
16    ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
17    {
18        /// <summary>
19        /// <para>
20        /// Initializes a new <see
21        ↪ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
22        /// </para>
23        /// <para></para>
24        /// </summary>
25        /// <param name="constants">
26        /// <para>A constants.</para>
27        /// <para></para>
28        /// </param>
29        /// <param name="linksDataParts">
30        /// <para>A links data parts.</para>
31        /// <para></para>
32        /// </param>
33        /// <param name="linksIndexParts">
34        /// <para>A links index parts.</para>
35        /// <para></para>
36        /// </param>
37        /// <param name="header">
38        /// <para>A header.</para>
39        /// <para></para>
40        /// </param>
41        [MethodImpl(MethodImplOptions.AggressiveInlining)]
42        public
43        ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
44        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
45        ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
46        ↪ linksIndexParts, header) { }
47
48        /// <summary>
49        /// <para>
50        /// Gets the left reference using the specified node.
51        /// </para>
52        /// <para></para>
53        /// </summary>
54        /// <param name="node">
55        /// <para>The node.</para>
56        /// <para></para>
57        /// </param>
58        /// <returns>
59        /// <para>The ref link</para>
60        /// <para></para>
61        /// </returns>
62        [MethodImpl(MethodImplOptions.AggressiveInlining)]
63        protected override ref TLink GetLeftReference(TLink node) => ref
64        ↪ LinksIndexParts[node].LeftAsTarget;
65
66        /// <summary>
67        /// <para>
68        /// Gets the right reference using the specified node.
69        /// </para>
70        /// <para></para>
71        /// </summary>
72        /// <param name="node">
73        /// <para>The node.</para>
74        /// <para></para>
75        /// </param>
76        /// <returns>
77        /// <para>The ref link</para>
78        /// <para></para>
79        /// </returns>
80        [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

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

```



```

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

```

```

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

```

1.59 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethod

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt32;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.Data.Doublets.Memory.Split.Specific
7 {
8     /// <summary>
9     /// <para>
10    /// Represents the int 32 internal links targets size balanced tree methods.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
15    public unsafe class UInt32InternalLinksTargetsSizeBalancedTreeMethods :
    ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt32InternalLinksTargetsSizeBalancedTreeMethods"/>
    ↪ instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="linksDataParts">
28        /// <para>A links data parts.</para>
29        /// <para></para>
30        /// </param>

```

```

31    /// <param name="linksIndexParts">
32    /// <para>A links index parts.</para>
33    /// <para></para>
34    /// </param>
35    /// <param name="header">
36    /// <para>A header.</para>
37    /// <para></para>
38    /// </param>
39    [MethodImpl(MethodImplOptions.AggressiveInlining)]
40    public UInt32InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
    ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
    ↪ linksIndexParts, header) { }

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

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

75
76    /// <summary>
77    /// <para>
78    /// Gets the left using the specified node.
79    /// </para>
80    /// <para></para>
81    /// </summary>
82    /// <param name="node">
83    /// <para>The node.</para>
84    /// <para></para>
85    /// </param>
86    /// <returns>
87    /// <para>The link</para>
88    /// <para></para>
89    /// </returns>
90    [MethodImpl(MethodImplOptions.AggressiveInlining)]
91    protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93    /// <summary>
94    /// <para>
95    /// Gets the right using the specified node.
96    /// </para>
97    /// <para></para>
98    /// </summary>
99    /// <param name="node">
100    /// <para>The node.</para>
101    /// <para></para>
102    /// </param>
103    /// <returns>

```

```

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

```

```

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

```

```

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

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLink = System.UInt32;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the int 32 split memory links.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="SplitMemoryLinksBase{TLink}" />
19     public unsafe class UInt32SplitMemoryLinks : SplitMemoryLinksBase<TLink>
20     {
21         /// <summary>
22         /// <para>
23         /// The create internal source tree methods.
24         /// </para>
25         /// <para></para>
26         /// </summary>
27         private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
28         /// <summary>
29         /// <para>
30         /// The create external source tree methods.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
35         /// <summary>
36         /// <para>
37         /// The create internal target tree methods.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
42         /// <summary>
43         /// <para>
44         /// The create external target tree methods.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
49         /// <summary>
50         /// <para>
51         /// The header.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         private LinksHeader<TLink>* _header;
56         /// <summary>
57         /// <para>
58         /// The links data parts.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         private RawLinkDataPart<TLink>* _linksDataParts;
63         /// <summary>
64         /// <para>

```

```

65     /// The links index parts.
66     /// </para>
67     /// <para></para>
68     /// </summary>
69     private RawLinkIndexPart<TLink>* _linksIndexParts;
70
71     /// <summary>
72     /// <para>
73     /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="dataMemory">
78     /// <para>A data memory.</para>
79     /// <para></para>
80     /// </param>
81     /// <param name="indexMemory">
82     /// <para>A index memory.</para>
83     /// <para></para>
84     /// </param>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
87         ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
88
89     /// <summary>
90     /// <para>
91     /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="dataMemory">
96     /// <para>A data memory.</para>
97     /// <para></para>
98     /// </param>
99     /// <param name="indexMemory">
100    /// <para>A index memory.</para>
101    /// <para></para>
102    /// </param>
103    /// <param name="memoryReservationStep">
104    /// <para>A memory reservation step.</para>
105    /// <para></para>
106    /// </param>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
109        ↪ indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
110        ↪ memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
111        ↪ IndexTreeType.Default, useLinkedList: true) { }
112
113    /// <summary>
114    /// <para>
115    /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
116    /// </para>
117    /// <para></para>
118    /// </summary>
119    /// <param name="dataMemory">
120    /// <para>A data memory.</para>
121    /// <para></para>
122    /// </param>
123    /// <param name="indexMemory">
124    /// <para>A index memory.</para>
125    /// <para></para>
126    /// </param>
127    /// <param name="memoryReservationStep">
128    /// <para>A memory reservation step.</para>
129    /// <para></para>
130    /// </param>
131    /// <param name="constants">
132    /// <para>A constants.</para>
133    /// <para></para>
134    /// </param>
135    [MethodImpl(MethodImplOptions.AggressiveInlining)]
136    public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
137        ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
138        ↪ this(dataMemory, indexMemory, memoryReservationStep, constants,
139        ↪ IndexTreeType.Default, useLinkedList: true) { }
140
141    /// <summary>
142    /// <para>

```

```

136 /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
137 /// </para>
138 /// </summary>
139 /// <param name="dataMemory">
140 /// <para>A data memory.</para>
141 /// </param>
142 /// <param name="indexMemory">
143 /// <para>A index memory.</para>
144 /// </param>
145 /// <param name="memoryReservationStep">
146 /// <para>A memory reservation step.</para>
147 /// </param>
148 /// <param name="constants">
149 /// <para>A constants.</para>
150 /// </param>
151 /// <param name="indexTreeType">
152 /// <para>A index tree type.</para>
153 /// </param>
154 /// <param name="useLinkedList">
155 /// <para>A use linked list.</para>
156 /// </param>
157 [MethodImpl(MethodImplOptions.AggressiveInlining)]
158 public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
159     ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
160     ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
161     ↪ memoryReservationStep, constants, useLinkedList)
162 {
163     if (indexTreeType == IndexTreeType.SizeBalancedTree)
164     {
165         _createInternalSourceTreeMethods = () => new
166             ↪ UInt32InternalLinksSourcesSizeBalancedTreeMethods(Constants,
167             ↪ _linksDataParts, _linksIndexParts, _header);
168         _createExternalSourceTreeMethods = () => new
169             ↪ UInt32ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
170             ↪ _linksDataParts, _linksIndexParts, _header);
171         _createInternalTargetTreeMethods = () => new
172             ↪ UInt32InternalLinksTargetsSizeBalancedTreeMethods(Constants,
173             ↪ _linksDataParts, _linksIndexParts, _header);
174         _createExternalTargetTreeMethods = () => new
175             ↪ UInt32ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
176             ↪ _linksDataParts, _linksIndexParts, _header);
177     }
178     else
179     {
180         _createInternalSourceTreeMethods = () => new
181             ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
182             ↪ _linksDataParts, _linksIndexParts, _header);
183         _createExternalSourceTreeMethods = () => new
184             ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
185             ↪ _linksDataParts, _linksIndexParts, _header);
186         _createInternalTargetTreeMethods = () => new
187             ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
188             ↪ _linksDataParts, _linksIndexParts, _header);
189         _createExternalTargetTreeMethods = () => new
190             ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
191             ↪ _linksDataParts, _linksIndexParts, _header);
192     }
193     Init(dataMemory, indexMemory);
194 }
195
196 /// <summary>
197 /// <para>
198 /// <para>Sets the pointers using the specified data memory.
199 /// </para>
200 /// </summary>
201 /// <param name="dataMemory">
202 /// <para>The data memory.</para>
203 /// </param>

```



```

194 /// <param name="indexMemory">
195 /// <para>The index memory.</para>
196 /// </para>
197 /// </param>
198 [MethodImpl(MethodImplOptions.AggressiveInlining)]
199 protected override void SetPointers(IResizableDirectMemory dataMemory,
200     ↳ IResizableDirectMemory indexMemory)
201 {
202     _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
203     _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
204     _header = (LinksHeader<TLink>*)indexMemory.Pointer;
205     if (_useLinkedList)
206     {
207         InternalSourcesListMethods = new
208             ↳ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
209                 ↳ _linksIndexParts);
210     }
211     else
212     {
213         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
214     }
215     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
216     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
217     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
218     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
219 }
220
221 /// <summary>
222 /// <para>
223 /// Resets the pointers.
224 /// </para>
225 /// </summary>
226 [MethodImpl(MethodImplOptions.AggressiveInlining)]
227 protected override void ResetPointers()
228 {
229     base.ResetPointers();
230     _linksDataParts = null;
231     _linksIndexParts = null;
232     _header = null;
233 }
234
235 /// <summary>
236 /// <para>
237 /// Gets the header reference.
238 /// </para>
239 /// </summary>
240 /// <returns>
241 /// <para>A ref links header of t link</para>
242 /// </returns>
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
245
246 /// <summary>
247 /// <para>
248 /// Gets the link data part reference using the specified link index.
249 /// </para>
250 /// </summary>
251 /// <param name="linkIndex">
252 /// <para>The link index.</para>
253 /// </para>
254 /// </param>
255 /// <returns>
256 /// <para>A ref raw link data part of t link</para>
257 /// </returns>
258 [MethodImpl(MethodImplOptions.AggressiveInlining)]
259 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
260     ↳ => ref _linksDataParts[linkIndex];
261
262 /// <summary>
263 /// <para>
264 /// Gets the link index part reference using the specified link index.
265 /// </para>
266 /// </summary>
267

```

```

268     /// </summary>
269     /// <param name="linkIndex">
270     /// <para>The link index.</para>
271     /// <para></para>
272     /// </param>
273     /// <returns>
274     /// <para>A ref raw link index part of t link</para>
275     /// <para></para>
276     /// </returns>
277     [MethodImpl(MethodImplOptions.AggressiveInlining)]
278     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
    ↪ linkIndex) => ref _linksIndexParts[linkIndex];

279     /// <summary>
280     /// <para>
281     /// Determines whether this instance are equal.
282     /// </para>
283     /// <para></para>
284     /// </summary>
285     /// <param name="first">
286     /// <para>The first.</para>
287     /// <para></para>
288     /// </param>
289     /// <param name="second">
290     /// <para>The second.</para>
291     /// <para></para>
292     /// </param>
293     /// <returns>
294     /// <para>The bool</para>
295     /// <para></para>
296     /// </returns>
297     [MethodImpl(MethodImplOptions.AggressiveInlining)]
298     protected override bool AreEqual(TLink first, TLink second) => first == second;

299     /// <summary>
300     /// <para>
301     /// Determines whether this instance less than.
302     /// </para>
303     /// <para></para>
304     /// </summary>
305     /// <param name="first">
306     /// <para>The first.</para>
307     /// <para></para>
308     /// </param>
309     /// <param name="second">
310     /// <para>The second.</para>
311     /// <para></para>
312     /// </param>
313     /// <returns>
314     /// <para>The bool</para>
315     /// <para></para>
316     /// </returns>
317     [MethodImpl(MethodImplOptions.AggressiveInlining)]
318     protected override bool LessThan(TLink first, TLink second) => first < second;

319     /// <summary>
320     /// <para>
321     /// Determines whether this instance less or equal than.
322     /// </para>
323     /// <para></para>
324     /// </summary>
325     /// <param name="first">
326     /// <para>The first.</para>
327     /// <para></para>
328     /// </param>
329     /// <param name="second">
330     /// <para>The second.</para>
331     /// <para></para>
332     /// </param>
333     /// <returns>
334     /// <para>The bool</para>
335     /// <para></para>
336     /// </returns>
337     [MethodImpl(MethodImplOptions.AggressiveInlining)]
338     protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;

339     /// <summary>
340     /// <para>

```

```

345     /// Determines whether this instance greater than.
346     /// </para>
347     /// <para></para>
348     /// </summary>
349     /// <param name="first">
350     /// <para>The first.</para>
351     /// <para></para>
352     /// </param>
353     /// <param name="second">
354     /// <para>The second.</para>
355     /// <para></para>
356     /// </param>
357     /// <returns>
358     /// <para>The bool</para>
359     /// <para></para>
360     /// </returns>
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     protected override bool GreaterThan(TLink first, TLink second) => first > second;
363
364     /// <summary>
365     /// <para>
366     /// Determines whether this instance greater or equal than.
367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="first">
371     /// <para>The first.</para>
372     /// <para></para>
373     /// </param>
374     /// <param name="second">
375     /// <para>The second.</para>
376     /// <para></para>
377     /// </param>
378     /// <returns>
379     /// <para>The bool</para>
380     /// <para></para>
381     /// </returns>
382     [MethodImpl(MethodImplOptions.AggressiveInlining)]
383     protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;
384
385     /// <summary>
386     /// <para>
387     /// Gets the zero.
388     /// </para>
389     /// <para></para>
390     /// </summary>
391     /// <returns>
392     /// <para>The link</para>
393     /// <para></para>
394     /// </returns>
395     [MethodImpl(MethodImplOptions.AggressiveInlining)]
396     protected override TLink GetZero() => 0U;
397
398     /// <summary>
399     /// <para>
400     /// Gets the one.
401     /// </para>
402     /// <para></para>
403     /// </summary>
404     /// <returns>
405     /// <para>The link</para>
406     /// <para></para>
407     /// </returns>
408     [MethodImpl(MethodImplOptions.AggressiveInlining)]
409     protected override TLink GetOne() => 1U;
410
411     /// <summary>
412     /// <para>
413     /// Converts the to int 64 using the specified value.
414     /// </para>
415     /// <para></para>
416     /// </summary>
417     /// <param name="value">
418     /// <para>The value.</para>
419     /// <para></para>
420     /// </param>
421     /// <returns>
422     /// <para>The long</para>

```

```

423     /// <para></para>
424     /// </returns>
425     [MethodImpl(MethodImplOptions.AggressiveInlining)]
426     protected override long ConvertToInt64(TLink value) => value;
427
428     /// <summary>
429     /// <para>
430     /// Converts the to address using the specified value.
431     /// </para>
432     /// <para></para>
433     /// </summary>
434     /// <param name="value">
435     /// <para>The value.</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 ConvertToAddress(long value) => (TLink)value;
444
445     /// <summary>
446     /// <para>
447     /// Adds the first.
448     /// </para>
449     /// <para></para>
450     /// </summary>
451     /// <param name="first">
452     /// <para>The first.</para>
453     /// <para></para>
454     /// </param>
455     /// <param name="second">
456     /// <para>The second.</para>
457     /// <para></para>
458     /// </param>
459     /// <returns>
460     /// <para>The link</para>
461     /// <para></para>
462     /// </returns>
463     [MethodImpl(MethodImplOptions.AggressiveInlining)]
464     protected override TLink Add(TLink first, TLink second) => first + second;
465
466     /// <summary>
467     /// <para>
468     /// Subtracts the first.
469     /// </para>
470     /// <para></para>
471     /// </summary>
472     /// <param name="first">
473     /// <para>The first.</para>
474     /// <para></para>
475     /// </param>
476     /// <param name="second">
477     /// <para>The second.</para>
478     /// <para></para>
479     /// </param>
480     /// <returns>
481     /// <para>The link</para>
482     /// <para></para>
483     /// </returns>
484     [MethodImpl(MethodImplOptions.AggressiveInlining)]
485     protected override TLink Subtract(TLink first, TLink second) => first - second;
486
487     /// <summary>
488     /// <para>
489     /// Increments the link.
490     /// </para>
491     /// <para></para>
492     /// </summary>
493     /// <param name="link">
494     /// <para>The link.</para>
495     /// <para></para>
496     /// </param>
497     /// <returns>
498     /// <para>The link</para>
499     /// <para></para>
500     /// </returns>

```

```

501     [MethodImpl(MethodImplOptions.AggressiveInlining)]
502     protected override TLink Increment(TLink link) => ++link;
503
504     /// <summary>
505     /// <para>
506     /// Decrements the link.
507     /// </para>
508     /// <para></para>
509     /// </summary>
510     /// <param name="link">
511     /// <para>The link.</para>
512     /// <para></para>
513     /// </param>
514     /// <returns>
515     /// <para>The link</para>
516     /// <para></para>
517     /// </returns>
518     [MethodImpl(MethodImplOptions.AggressiveInlining)]
519     protected override TLink Decrement(TLink link) => --link;
520 }
521 }

```

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         /// <summary>
19         /// <para>
20         /// The links.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private readonly RawLinkDataPart<TLink>* _links;
25         /// <summary>
26         /// <para>
27         /// The header.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         private readonly LinksHeader<TLink>* _header;
32
33         /// <summary>
34         /// <para>
35         /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="links">
40         /// <para>A links.</para>
41         /// <para></para>
42         /// </param>
43         /// <param name="header">
44         /// <para>A header.</para>
45         /// <para></para>
46         /// </param>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public UInt32UnusedLinksListMethods(RawLinkDataPart<TLink>* links, LinksHeader<TLink>*
49             ↪ header)
50             : base((byte*)links, (byte*)header)
51         {
52             _links = links;
53             _header = header;
54         }
55
56         /// <summary>

```

```

56     /// <para>
57     /// Gets the link data part reference using the specified link.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="link">
62     /// <para>The link.</para>
63     /// <para></para>
64     /// </param>
65     /// <returns>
66     /// <para>A ref raw link data part of t link</para>
67     /// <para></para>
68     /// </returns>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
71         ↪ ref _links[link];
72
73     /// <summary>
74     /// <para>
75     /// Gets the header reference.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <returns>
80     /// <para>A ref links header of t link</para>
81     /// <para></para>
82     /// </returns>
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
85 }

```

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>

```

```

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     ↪ UInt64 ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(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     /// <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     /// <summary>
87     /// <para>
88     /// Determines whether this instance equal to zero.
89     /// </para>
90     /// <para></para>
91     /// </summary>
92     /// <param name="value">
93     /// <para>The value.</para>
94     /// <para></para>
95     /// </param>
96     /// <returns>
97     /// <para>The bool</para>
98     /// <para></para>
99     /// </returns>
100    [MethodImpl(MethodImplOptions.AggressiveInlining)]
101    protected override bool EqualToZero(ulong value) => value == OUL;
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance are equal.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="first">
109    /// <para>The first.</para>
110    /// <para></para>
111    /// </param>
112    /// <param name="second">
113    /// <para>The second.</para>
114    /// <para></para>
115    /// </param>
116    /// <returns>
117    /// <para>The bool</para>
118    /// <para></para>
119    /// </returns>

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

48     /// <para>A constants.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="linksDataParts">
52     /// <para>A links data parts.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="linksIndexParts">
56     /// <para>A links index parts.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="header">
60     /// <para>A header.</para>
61     /// <para></para>
62     /// </param>
63     [MethodImpl(MethodImplOptions.AggressiveInlining)]
64     protected UInt64ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header)
        : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65     {
66     LinksDataParts = linksDataParts;
67     LinksIndexParts = linksIndexParts;
68     Header = header;
69     }
70
71     /// <summary>
72     /// <para>
73     /// Gets the zero.
74     /// </para>
75     /// </summary>
76     /// <para></para>
77     /// </summary>
78     /// <returns>
79     /// <para>The ulong</para>
80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override ulong GetZero() => 0UL;
84
85     /// <summary>
86     /// <para>
87     /// Determines whether this instance equal to zero.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <param name="value">
92     /// <para>The value.</para>
93     /// <para></para>
94     /// </param>
95     /// <returns>
96     /// <para>The bool</para>
97     /// <para></para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    protected override bool EqualToZero(ulong value) => value == 0UL;
101
102    /// <summary>
103    /// <para>
104    /// Determines whether this instance are equal.
105    /// </para>
106    /// <para></para>
107    /// </summary>
108    /// <param name="first">
109    /// <para>The first.</para>
110    /// <para></para>
111    /// </param>
112    /// <param name="second">
113    /// <para>The second.</para>
114    /// <para></para>
115    /// </param>
116    /// <returns>
117    /// <para>The bool</para>
118    /// <para></para>
119    /// </returns>
120    [MethodImpl(MethodImplOptions.AggressiveInlining)]
121    protected override bool AreEqual(ulong first, ulong second) => first == second;
122
123    /// <summary>

```

```

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

```

```

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

```

```

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

```



```

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

```

```

430     /// <param name="first">
431     /// <para>The first.</para>
432     /// <para></para>
433     /// </param>
434     /// <param name="second">
435     /// <para>The second.</para>
436     /// <para></para>
437     /// </param>
438     /// <returns>
439     /// <para>The bool</para>
440     /// <para></para>
441     /// </returns>
442     [MethodImpl(MethodImplOptions.AggressiveInlining)]
443     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
444     {
445         ref var firstLink = ref LinksDataParts[first];
446         ref var secondLink = ref LinksDataParts[second];
447         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
448             ↪ secondLink.Source, secondLink.Target);
449     }
450 }

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

1.65 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods

```

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

```

```

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     ↪ 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">

```

```

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

```



```

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

```

```

284         link.SizeAsSource = Zero;
285     }
286 }
287 }

```

1.66 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethodsBase

```

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

```

```

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

```

```

142     protected override void SetRight(TLink node, TLink right) =>
143         ↳ LinksIndexParts[node].RightAsTarget = right;
144
145     /// <summary>
146     /// <para>
147     /// Gets the size using the specified node.
148     /// </para>
149     /// </summary>
150     /// <param name="node">
151     /// <para>The node.</para>
152     /// </param>
153     /// <returns>
154     /// <para>The link</para>
155     /// </returns>
156     [MethodImpl(MethodImplOptions.AggressiveInlining)]
157     protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsTarget;
158
159     /// <summary>
160     /// <para>
161     /// Sets the size using the specified node.
162     /// </para>
163     /// </summary>
164     /// <param name="node">
165     /// <para>The node.</para>
166     /// </param>
167     /// <param name="size">
168     /// <para>The size.</para>
169     /// </param>
170     [MethodImpl(MethodImplOptions.AggressiveInlining)]
171     protected override void SetSize(TLink node, TLink size) =>
172         ↳ LinksIndexParts[node].SizeAsTarget = size;
173
174     /// <summary>
175     /// <para>
176     /// Gets the tree root.
177     /// </para>
178     /// </summary>
179     /// <returns>
180     /// <para>The link</para>
181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     protected override TLink GetTreeRoot() => Header->RootAsTarget;
184
185     /// <summary>
186     /// <para>
187     /// Gets the base part value using the specified node.
188     /// </para>
189     /// </summary>
190     /// <param name="node">
191     /// <para>The node.</para>
192     /// </param>
193     /// <returns>
194     /// <para>The link</para>
195     /// </returns>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]
197     protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
198
199     /// <summary>
200     /// <para>
201     /// Determines whether this instance first is to the left of second.
202     /// </para>
203     /// </summary>
204     /// <param name="firstSource">
205     /// <para>The first source.</para>
206     /// </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;
271
272     /// <summary>
273     /// <para>
274     /// Clears the node using the specified node.
275     /// </para>
276     /// <para></para>
277     /// </summary>
278     /// <param name="node">
279     /// <para>The node.</para>
280     /// <para></para>
281     /// </param>
282     [MethodImpl(MethodImplOptions.AggressiveInlining)]
283     protected override void ClearNode(TLink node)
284     {
285         ref var link = ref LinksIndexParts[node];
286         link.LeftAsTarget = Zero;
287         link.RightAsTarget = Zero;
288         link.SizeAsTarget = Zero;
289     }
290 }
291 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using TLink = System.UInt64;

```

```

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

```

```

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

```

```

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.
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>

```



```

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

```

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 :
    ↳ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
17  {
18      /// <summary>
19      /// <para>
20      /// The links data parts.
21      /// </para>
22      /// <para></para>
23      /// </summary>
24      protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
25      /// <summary>
26      /// <para>
27      /// The links index parts.
28      /// </para>
29      /// <para></para>
30      /// </summary>
31      protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
32      /// <summary>
33      /// <para>
34      /// The header.
35      /// </para>
36      /// <para></para>
37      /// </summary>
38      protected new readonly LinksHeader<TLink>* Header;
39
40      /// <summary>
41      /// <para>
42      /// Initializes a new <see
    ↳ cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
43      /// </para>
44      /// <para></para>
45      /// </summary>
46      /// <param name="constants">
47      /// <para>A constants.</para>
48      /// <para></para>
49      /// </param>
50      /// <param name="linksDataParts">
51      /// <para>A links data parts.</para>
52      /// <para></para>
53      /// </param>
54      /// <param name="linksIndexParts">
55      /// <para>A links index parts.</para>
56      /// <para></para>
57      /// </param>
58      /// <param name="header">
59      /// <para>A header.</para>
60      /// <para></para>
61      /// </param>
62      [MethodImpl(MethodImplOptions.AggressiveInlining)]
63      protected
    ↳ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
    ↳ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
    ↳ linksIndexParts, LinksHeader<TLink>* header)
    ↳ : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
64      {
65          LinksDataParts = linksDataParts;
66          LinksIndexParts = linksIndexParts;
67          Header = header;
68      }
69
70      /// <summary>
71      /// <para>
72      /// Gets the zero.
73      /// </para>
74      /// <para></para>
75      /// </summary>
76      /// <returns>
77      /// <para>The ulong</para>
78      /// <para></para>
79      /// </returns>
80      [MethodImpl(MethodImplOptions.AggressiveInlining)]
81      protected override ulong GetZero() => 0UL;
82
83

```

```

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

```

```

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

```

```

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

```

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
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>

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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         /// <summary>
18         /// <para>
19         /// The links data parts.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         private readonly RawLinkDataPart<TLink>* _linksDataParts;
24         /// <summary>
25         /// <para>
26         /// The links index parts.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="UInt64InternalLinksSourcesLinkedListMethods" /> 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="linksDataParts">
43         /// <para>A links data parts.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="linksIndexParts">
47         /// <para>A links index parts.</para>
48         /// <para></para>
49         /// </param>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public UInt64InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
            ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
52         {
53             _linksDataParts = linksDataParts;
54             _linksIndexParts = linksIndexParts;
55         }
56
57         /// <summary>
58         /// <para>
59         /// Gets the link data part reference using the specified link.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <param name="link">
64         /// <para>The link.</para>
65         /// </param>

```

```

66     /// <para></para>
67     /// </param>
68     /// <returns>
69     /// <para>A ref raw link data part of t link</para>
70     /// <para></para>
71     /// </returns>
72     [MethodImpl(MethodImplOptions.AggressiveInlining)]
73     protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
74         ↪ ref _linksDataParts[link];
75
76     /// <summary>
77     /// <para>
78     /// Gets the link index part reference using the specified link.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     /// <param name="link">
83     /// <para>The link.</para>
84     /// <para></para>
85     /// </param>
86     /// <returns>
87     /// <para>A ref raw link index part of t link</para>
88     /// <para></para>
89     /// </returns>
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
92         ↪ ref _linksIndexParts[link];
93 }
94 }

```

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
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see
21         ↪ cref="UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="constants">
26         /// <para>A constants.</para>
27         /// <para></para>
28         /// </param>
29         /// <param name="linksDataParts">
30         /// <para>A links data parts.</para>
31         /// <para></para>
32         /// </param>
33         /// <param name="linksIndexParts">
34         /// <para>A links index parts.</para>
35         /// <para></para>
36         /// </param>
37         /// <param name="header">
38         /// <para>A header.</para>
39         /// <para></para>
40         /// </param>
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         public
43         ↪ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
44         ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
45         ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
46         ↪ linksIndexParts, header) { }
47     }
48 }

```

```

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)]

```



```

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

```

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

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```

```

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

```

```

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

```

```

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

```



```

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

```

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

```

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

```

```

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

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

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

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

```

```

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

```

```

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

```

1.75 ./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      /// <summary>
22      /// <para>
23      /// The create internal source tree methods.
24      /// </para>
25      /// <para></para>
26      /// </summary>
27      private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
28      /// <summary>
29      /// <para>
30      /// The create external source tree methods.
31      /// </para>
32      /// <para></para>
33      /// </summary>
34      private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
35      /// <summary>
36      /// <para>
37      /// The create internal target tree methods.
38      /// </para>
39      /// <para></para>
40      /// </summary>
41      private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
42      /// <summary>
43      /// <para>
44      /// The create external target tree methods.
45      /// </para>
46      /// <para></para>
47      /// </summary>
48      private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
49      /// <summary>
50      /// <para>
51      /// The header.
52      /// </para>
53      /// <para></para>
54      /// </summary>
55      private LinksHeader<ulong>* _header;
56      /// <summary>
57      /// <para>
58      /// The links data parts.
59      /// </para>
60      /// <para></para>
61      /// </summary>
62      private RawLinkDataPart<ulong>* _linksDataParts;
63      /// <summary>
64      /// <para>
65      /// The links index parts.
66      /// </para>
67      /// <para></para>
68      /// </summary>
69      private RawLinkIndexPart<ulong>* _linksIndexParts;
70
71      /// <summary>
72      /// <para>
73      /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
74      /// </para>
75      /// <para></para>
76      /// </summary>
77      /// <param name="dataMemory">
78      /// <para>A data memory.</para>
79      /// <para></para>
80      /// </param>
81      /// <param name="indexMemory">
82      /// <para>A index memory.</para>
83      /// <para></para>
84      /// </param>
85      [MethodImpl(MethodImplOptions.AggressiveInlining)]
86      public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
87      ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
88
89      /// <summary>
90      /// <para>
91      /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.

```

```

91    /// </para>
92    /// <para></para>
93    /// </summary>
94    /// <param name="dataMemory">
95    /// <para>A data memory.</para>
96    /// <para></para>
97    /// </param>
98    /// <param name="indexMemory">
99    /// <para>A index memory.</para>
100   /// <para></para>
101   /// </param>
102   /// <param name="memoryReservationStep">
103   /// <para>A memory reservation step.</para>
104   /// <para></para>
105   /// </param>
106   [MethodImpl(MethodImplOptions.AggressiveInlining)]
107   public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    → indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
    → memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
    → IndexTreeType.Default, useLinkedList: true) { }

108
109   /// <summary>
110   /// <para>
111   /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
112   /// </para>
113   /// <para></para>
114   /// </summary>
115   /// <param name="dataMemory">
116   /// <para>A data memory.</para>
117   /// <para></para>
118   /// </param>
119   /// <param name="indexMemory">
120   /// <para>A index memory.</para>
121   /// <para></para>
122   /// </param>
123   /// <param name="memoryReservationStep">
124   /// <para>A memory reservation step.</para>
125   /// <para></para>
126   /// </param>
127   /// <param name="constants">
128   /// <para>A constants.</para>
129   /// <para></para>
130   /// </param>
131   [MethodImpl(MethodImplOptions.AggressiveInlining)]
132   public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    → indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
    → this(dataMemory, indexMemory, memoryReservationStep, constants,
    → IndexTreeType.Default, useLinkedList: true) { }

133
134   /// <summary>
135   /// <para>
136   /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
137   /// </para>
138   /// <para></para>
139   /// </summary>
140   /// <param name="dataMemory">
141   /// <para>A data memory.</para>
142   /// <para></para>
143   /// </param>
144   /// <param name="indexMemory">
145   /// <para>A index memory.</para>
146   /// <para></para>
147   /// </param>
148   /// <param name="memoryReservationStep">
149   /// <para>A memory reservation step.</para>
150   /// <para></para>
151   /// </param>
152   /// <param name="constants">
153   /// <para>A constants.</para>
154   /// <para></para>
155   /// </param>
156   /// <param name="indexTreeType">
157   /// <para>A index tree type.</para>
158   /// <para></para>
159   /// </param>
160   /// <param name="useLinkedList">
161   /// <para>A use linked list.</para>

```

```

162 /// <para></para>
163 /// </param>
164 [MethodImpl(MethodImplOptions.AggressiveInlining)]
165 public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
    ↳ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
    ↳ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
    ↳ memoryReservationStep, constants, useLinkedList)
166 {
167     if (indexTreeType == IndexTreeType.SizeBalancedTree)
168     {
169         _createInternalSourceTreeMethods = () => new
            ↳ UInt64InternalLinksSourcesSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
170         _createExternalSourceTreeMethods = () => new
            ↳ UInt64ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
171         _createInternalTargetTreeMethods = () => new
            ↳ UInt64InternalLinksTargetsSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
172         _createExternalTargetTreeMethods = () => new
            ↳ UInt64ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
173     }
174     else
175     {
176         _createInternalSourceTreeMethods = () => new
            ↳ UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
177         _createExternalSourceTreeMethods = () => new
            ↳ UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
178         _createInternalTargetTreeMethods = () => new
            ↳ UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
179         _createExternalTargetTreeMethods = () => new
            ↳ UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
            ↳ _linksDataParts, _linksIndexParts, _header);
180     }
181     Init(dataMemory, indexMemory);
182 }
183
184 /// <summary>
185 /// <para>
186 /// Sets the pointers using the specified data memory.
187 /// </para>
188 /// <para></para>
189 /// </summary>
190 /// <param name="dataMemory">
191 /// <para>The data memory.</para>
192 /// <para></para>
193 /// </param>
194 /// <param name="indexMemory">
195 /// <para>The index memory.</para>
196 /// <para></para>
197 /// </param>
198 [MethodImpl(MethodImplOptions.AggressiveInlining)]
199 protected override void SetPointers(IResizableDirectMemory dataMemory,
    ↳ IResizableDirectMemory indexMemory)
200 {
201     _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
202     _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
203     _header = (LinksHeader<TLink>*)indexMemory.Pointer;
204     if (_useLinkedList)
205     {
206         InternalSourcesListMethods = new
            ↳ UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
            ↳ _linksIndexParts);
207     }
208     else
209     {
210         InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
211     }
212     ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
213     InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
214     ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
215     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_linksDataParts, _header);
216 }

```

```

217
218 /// <summary>
219 /// <para>
220 /// Resets the pointers.
221 /// </para>
222 /// <para></para>
223 /// </summary>
224 [MethodImpl(MethodImplOptions.AggressiveInlining)]
225 protected override void ResetPointers()
226 {
227     base.ResetPointers();
228     _linksDataParts = null;
229     _linksIndexParts = null;
230     _header = null;
231 }
232
233 /// <summary>
234 /// <para>
235 /// Gets the header reference.
236 /// </para>
237 /// <para></para>
238 /// </summary>
239 /// <returns>
240 /// <para>A ref links header of t link</para>
241 /// <para></para>
242 /// </returns>
243 [MethodImpl(MethodImplOptions.AggressiveInlining)]
244 protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
245
246 /// <summary>
247 /// <para>
248 /// Gets the link data part reference using the specified link index.
249 /// </para>
250 /// <para></para>
251 /// </summary>
252 /// <param name="linkIndex">
253 /// <para>The link index.</para>
254 /// <para></para>
255 /// </param>
256 /// <returns>
257 /// <para>A ref raw link data part of t link</para>
258 /// <para></para>
259 /// </returns>
260 [MethodImpl(MethodImplOptions.AggressiveInlining)]
261 protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
262     => ref _linksDataParts[linkIndex];
263
264 /// <summary>
265 /// <para>
266 /// Gets the link index part reference using the specified link index.
267 /// </para>
268 /// <para></para>
269 /// </summary>
270 /// <param name="linkIndex">
271 /// <para>The link index.</para>
272 /// <para></para>
273 /// </param>
274 /// <returns>
275 /// <para>A ref raw link index part of t link</para>
276 /// <para></para>
277 /// </returns>
278 [MethodImpl(MethodImplOptions.AggressiveInlining)]
279 protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
280     linkIndex) => ref _linksIndexParts[linkIndex];
281
282 /// <summary>
283 /// <para>
284 /// Determines whether this instance are equal.
285 /// </para>
286 /// <para></para>
287 /// </summary>
288 /// <param name="first">
289 /// <para>The first.</para>
290 /// <para></para>
291 /// </param>
292 /// <param name="second">
293 /// <para>The second.</para>
294 /// <para></para>
295 /// </param>

```



```

293     /// </param>
294     /// <returns>
295     /// <para>The bool</para>
296     /// <para></para>
297     /// </returns>
298     [MethodImpl(MethodImplOptions.AggressiveInlining)]
299     protected override bool AreEqual(ulong first, ulong second) => first == second;
300
301     /// <summary>
302     /// <para>
303     /// Determines whether this instance less 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 LessThan(ulong first, ulong second) => first < second;
321
322     /// <summary>
323     /// <para>
324     /// Determines whether this instance less 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 LessOrEqualThan(ulong first, ulong second) => first <= second;
342
343     /// <summary>
344     /// <para>
345     /// Determines whether this instance greater than.
346     /// </para>
347     /// <para></para>
348     /// </summary>
349     /// <param name="first">
350     /// <para>The first.</para>
351     /// <para></para>
352     /// </param>
353     /// <param name="second">
354     /// <para>The second.</para>
355     /// <para></para>
356     /// </param>
357     /// <returns>
358     /// <para>The bool</para>
359     /// <para></para>
360     /// </returns>
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     protected override bool GreaterThan(ulong first, ulong second) => first > second;
363
364     /// <summary>
365     /// <para>
366     /// Determines whether this instance greater or equal than.
367     /// </para>
368     /// <para></para>
369     /// </summary>
370     /// <param name="first">

```

```

371     /// <para>The first.</para>
372     /// <para></para>
373     /// </param>
374     /// <param name="second">
375     /// <para>The second.</para>
376     /// <para></para>
377     /// </param>
378     /// <returns>
379     /// <para>The bool</para>
380     /// <para></para>
381     /// </returns>
382     [MethodImpl(MethodImplOptions.AggressiveInlining)]
383     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
384
385     /// <summary>
386     /// <para>
387     /// Gets the zero.
388     /// </para>
389     /// <para></para>
390     /// </summary>
391     /// <returns>
392     /// <para>The ulong</para>
393     /// <para></para>
394     /// </returns>
395     [MethodImpl(MethodImplOptions.AggressiveInlining)]
396     protected override ulong GetZero() => 0UL;
397
398     /// <summary>
399     /// <para>
400     /// Gets the one.
401     /// </para>
402     /// <para></para>
403     /// </summary>
404     /// <returns>
405     /// <para>The ulong</para>
406     /// <para></para>
407     /// </returns>
408     [MethodImpl(MethodImplOptions.AggressiveInlining)]
409     protected override ulong GetOne() => 1UL;
410
411     /// <summary>
412     /// <para>
413     /// Converts the to int 64 using the specified value.
414     /// </para>
415     /// <para></para>
416     /// </summary>
417     /// <param name="value">
418     /// <para>The value.</para>
419     /// <para></para>
420     /// </param>
421     /// <returns>
422     /// <para>The long</para>
423     /// <para></para>
424     /// </returns>
425     [MethodImpl(MethodImplOptions.AggressiveInlining)]
426     protected override long ConvertToInt64(ulong value) => (long)value;
427
428     /// <summary>
429     /// <para>
430     /// Converts the to address using the specified value.
431     /// </para>
432     /// <para></para>
433     /// </summary>
434     /// <param name="value">
435     /// <para>The value.</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 ConvertToAddress(long value) => (ulong)value;
444
445     /// <summary>
446     /// <para>
447     /// Adds the first.
448     /// </para>

```

```

449     /// <para></para>
450     /// </summary>
451     /// <param name="first">
452     /// <para>The first.</para>
453     /// <para></para>
454     /// </param>
455     /// <param name="second">
456     /// <para>The second.</para>
457     /// <para></para>
458     /// </param>
459     /// <returns>
460     /// <para>The ulong</para>
461     /// <para></para>
462     /// </returns>
463     [MethodImpl(MethodImplOptions.AggressiveInlining)]
464     protected override ulong Add(ulong first, ulong second) => first + second;
465
466     /// <summary>
467     /// <para>
468     /// Subtracts the first.
469     /// </para>
470     /// <para></para>
471     /// </summary>
472     /// <param name="first">
473     /// <para>The first.</para>
474     /// <para></para>
475     /// </param>
476     /// <param name="second">
477     /// <para>The second.</para>
478     /// <para></para>
479     /// </param>
480     /// <returns>
481     /// <para>The ulong</para>
482     /// <para></para>
483     /// </returns>
484     [MethodImpl(MethodImplOptions.AggressiveInlining)]
485     protected override ulong Subtract(ulong first, ulong second) => first - second;
486
487     /// <summary>
488     /// <para>
489     /// Increments the link.
490     /// </para>
491     /// <para></para>
492     /// </summary>
493     /// <param name="link">
494     /// <para>The link.</para>
495     /// <para></para>
496     /// </param>
497     /// <returns>
498     /// <para>The ulong</para>
499     /// <para></para>
500     /// </returns>
501     [MethodImpl(MethodImplOptions.AggressiveInlining)]
502     protected override ulong Increment(ulong link) => ++link;
503
504     /// <summary>
505     /// <para>
506     /// Decrements the link.
507     /// </para>
508     /// <para></para>
509     /// </summary>
510     /// <param name="link">
511     /// <para>The link.</para>
512     /// <para></para>
513     /// </param>
514     /// <returns>
515     /// <para>The ulong</para>
516     /// <para></para>
517     /// </returns>
518     [MethodImpl(MethodImplOptions.AggressiveInlining)]
519     protected override ulong Decrement(ulong link) => --link;
520 }
521 }

```

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        /// <summary>
19        /// <para>
20        /// The links.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        private readonly RawLinkDataPart<ulong>* _links;
25        /// <summary>
26        /// <para>
27        /// The header.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        private readonly LinksHeader<ulong>* _header;
32
33        /// <summary>
34        /// <para>
35        /// Initializes a new <see cref="UInt64UnusedLinksListMethods"/> instance.
36        /// </para>
37        /// <para></para>
38        /// </summary>
39        /// <param name="links">
40        /// <para>A links.</para>
41        /// <para></para>
42        /// </param>
43        /// <param name="header">
44        /// <para>A header.</para>
45        /// <para></para>
46        /// </param>
47        [MethodImpl(MethodImplOptions.AggressiveInlining)]
48        public UInt64UnusedLinksListMethods(RawLinkDataPart<ulong>* links, LinksHeader<ulong>*
49        → header)
50        : base((byte*)links, (byte*)header)
51        {
52            _links = links;
53            _header = header;
54        }
55
56        /// <summary>
57        /// <para>
58        /// Gets the link data part reference using the specified link.
59        /// </para>
60        /// <para></para>
61        /// </summary>
62        /// <param name="link">
63        /// <para>The link.</para>
64        /// <para></para>
65        /// </param>
66        /// <returns>
67        /// <para>A ref raw link data part of t link</para>
68        /// <para></para>
69        /// </returns>
70        [MethodImpl(MethodImplOptions.AggressiveInlining)]
71        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
72        → ref _links[link];
73
74        /// <summary>
75        /// <para>
76        /// Gets the header reference.
77        /// </para>
78        /// <para></para>
79        /// </summary>
80        /// <returns>
81        /// <para>A ref links header of t link</para>

```

```

80     /// <para></para>
81     /// </returns>
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
84 }
85 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Collections.Methods.Trees;
6  using Platform.Converters;
7  using Platform.Numbers;
8  using static System.Runtime.CompilerServices.Unsafe;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Memory.United.Generic
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the links avl balanced tree methods base.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="SizedAndThreadedAVLBalancedTreeMethods{TLink}" />
21     /// <seealso cref="ILinksTreeMethods{TLink}" />
22     public unsafe abstract class LinksAvlBalancedTreeMethodsBase<TLink> :
23     ↪ SizedAndThreadedAVLBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
24     {
25         /// <summary>
26         /// <para>
27         /// The default.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
32         ↪ UncheckedConverter<TLink, long>.Default;
33         /// <summary>
34         /// <para>
35         /// The default.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         private static readonly UncheckedConverter<TLink, int> _addressToInt32Converter =
40         ↪ UncheckedConverter<TLink, int>.Default;
41         /// <summary>
42         /// <para>
43         /// The default.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         private static readonly UncheckedConverter<bool, TLink> _boolToAddressConverter =
48         ↪ UncheckedConverter<bool, TLink>.Default;
49         /// <summary>
50         /// <para>
51         /// The default.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         private static readonly UncheckedConverter<TLink, bool> _addressToBoolConverter =
56         ↪ UncheckedConverter<TLink, bool>.Default;
57         /// <summary>
58         /// <para>
59         /// The default.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         private static readonly UncheckedConverter<int, TLink> _int32ToAddressConverter =
64         ↪ UncheckedConverter<int, TLink>.Default;
65
66         /// <summary>
67         /// <para>
68         /// The break.
69         /// </para>
70         /// <para></para>
71         /// </summary>

```

```

66     protected readonly TLink Break;
67     /// <summary>
68     /// <para>
69     /// The continue.
70     /// </para>
71     /// <para></para>
72     /// </summary>
73     protected readonly TLink Continue;
74     /// <summary>
75     /// <para>
76     /// The links.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     protected readonly byte* Links;
81     /// <summary>
82     /// <para>
83     /// The header.
84     /// </para>
85     /// <para></para>
86     /// </summary>
87     protected readonly byte* Header;
88
89     /// <summary>
90     /// <para>
91     /// Initializes a new <see cref="LinksAvlBalancedTreeMethodsBase"/> instance.
92     /// </para>
93     /// <para></para>
94     /// </summary>
95     /// <param name="constants">
96     /// <para>A constants.</para>
97     /// <para></para>
98     /// </param>
99     /// <param name="links">
100    /// <para>A links.</para>
101    /// <para></para>
102    /// </param>
103    /// <param name="header">
104    /// <para>A header.</para>
105    /// <para></para>
106    /// </param>
107    [MethodImpl(MethodImplOptions.AggressiveInlining)]
108    protected LinksAvlBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
109        ↪ byte* header)
110    {
111        Links = links;
112        Header = header;
113        Break = constants.Break;
114        Continue = constants.Continue;
115    }
116
117    /// <summary>
118    /// <para>
119    /// Gets the tree root.
120    /// </para>
121    /// <para></para>
122    /// </summary>
123    /// <returns>
124    /// <para>The link</para>
125    /// <para></para>
126    /// </returns>
127    [MethodImpl(MethodImplOptions.AggressiveInlining)]
128    protected abstract TLink GetTreeRoot();
129
130    /// <summary>
131    /// <para>
132    /// Gets the base part value using the specified link.
133    /// </para>
134    /// <para></para>
135    /// </summary>
136    /// <param name="link">
137    /// <para>The link.</para>
138    /// <para></para>
139    /// </param>
140    /// <returns>
141    /// <para>The link</para>
142    /// <para></para>
143    /// </returns>

```

```

143 [MethodImpl(MethodImplOptions.AggressiveInlining)]
144 protected abstract TLink GetBasePartValue(TLink link);
145
146 /// <summary>
147 /// <para>
148 /// Determines whether this instance first is to the right of second.
149 /// </para>
150 /// <para></para>
151 /// </summary>
152 /// <param name="source">
153 /// <para>The source.</para>
154 /// <para></para>
155 /// </param>
156 /// <param name="target">
157 /// <para>The target.</para>
158 /// <para></para>
159 /// </param>
160 /// <param name="rootSource">
161 /// <para>The root source.</para>
162 /// <para></para>
163 /// </param>
164 /// <param name="rootTarget">
165 /// <para>The root target.</para>
166 /// <para></para>
167 /// </param>
168 /// <returns>
169 /// <para>The bool</para>
170 /// <para></para>
171 /// </returns>
172 [MethodImpl(MethodImplOptions.AggressiveInlining)]
173 protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);
174
175 /// <summary>
176 /// <para>
177 /// Determines whether this instance first is to the left of second.
178 /// </para>
179 /// <para></para>
180 /// </summary>
181 /// <param name="source">
182 /// <para>The source.</para>
183 /// <para></para>
184 /// </param>
185 /// <param name="target">
186 /// <para>The target.</para>
187 /// <para></para>
188 /// </param>
189 /// <param name="rootSource">
190 /// <para>The root source.</para>
191 /// <para></para>
192 /// </param>
193 /// <param name="rootTarget">
194 /// <para>The root target.</para>
195 /// <para></para>
196 /// </param>
197 /// <returns>
198 /// <para>The bool</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
    ↪ rootSource, TLink rootTarget);
203
204 /// <summary>
205 /// <para>
206 /// Gets the header reference.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 /// <returns>
211 /// <para>A ref links header of t link</para>
212 /// <para></para>
213 /// </returns>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
    ↪ AsRef<LinksHeader<TLink>>(Header);
216
217 /// <summary>

```

```

218 /// <para>
219 /// Gets the link reference using the specified link.
220 /// </para>
221 /// <para></para>
222 /// </summary>
223 /// <param name="link">
224 /// <para>The link.</para>
225 /// <para></para>
226 /// </param>
227 /// <returns>
228 /// <para>A ref raw link of t link</para>
229 /// <para></para>
230 /// </returns>
231 [MethodImpl(MethodImplOptions.AggressiveInlining)]
232 protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
    ↳ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
    ↳ _addressToInt64Converter.Convert(link)));
233
234 /// <summary>
235 /// <para>
236 /// Gets the link values using the specified link index.
237 /// </para>
238 /// <para></para>
239 /// </summary>
240 /// <param name="linkIndex">
241 /// <para>The link index.</para>
242 /// <para></para>
243 /// </param>
244 /// <returns>
245 /// <para>A list of t link</para>
246 /// <para></para>
247 /// </returns>
248 [MethodImpl(MethodImplOptions.AggressiveInlining)]
249 protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
250 {
251     ref var link = ref GetLinkReference(linkIndex);
252     return new Link<TLink>(linkIndex, link.Source, link.Target);
253 }
254
255 /// <summary>
256 /// <para>
257 /// Determines whether this instance first is to the left of second.
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 FirstIsToLeftOfSecond(TLink first, TLink second)
275 {
276     ref var firstLink = ref GetLinkReference(first);
277     ref var secondLink = ref GetLinkReference(second);
278     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
    ↳ secondLink.Source, secondLink.Target);
279 }
280
281 /// <summary>
282 /// <para>
283 /// Determines whether this instance first is to the right of second.
284 /// </para>
285 /// <para></para>
286 /// </summary>
287 /// <param name="first">
288 /// <para>The first.</para>
289 /// <para></para>
290 /// </param>
291 /// <param name="second">
292 /// <para>The second.</para>

```



```

293     /// <para></para>
294     /// </param>
295     /// <returns>
296     /// <para>The bool</para>
297     /// <para></para>
298     /// </returns>
299     [MethodImpl(MethodImplOptions.AggressiveInlining)]
300     protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
301     {
302         ref var firstLink = ref GetLinkReference(first);
303         ref var secondLink = ref GetLinkReference(second);
304         return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
305             ↪ secondLink.Source, secondLink.Target);
306     }
307     /// <summary>
308     /// <para>
309     /// Gets the size value using the specified value.
310     /// </para>
311     /// <para></para>
312     /// </summary>
313     /// <param name="value">
314     /// <para>The value.</para>
315     /// <para></para>
316     /// </param>
317     /// <returns>
318     /// <para>The link</para>
319     /// <para></para>
320     /// </returns>
321     [MethodImpl(MethodImplOptions.AggressiveInlining)]
322     protected virtual TLink GetSizeValue(TLink value) => Bit<TLink>.PartialRead(value, 5,
323         ↪ -5);
324     /// <summary>
325     /// <para>
326     /// Sets the size value using the specified stored value.
327     /// </para>
328     /// <para></para>
329     /// </summary>
330     /// <param name="storedValue">
331     /// <para>The stored value.</para>
332     /// <para></para>
333     /// </param>
334     /// <param name="size">
335     /// <para>The size.</para>
336     /// <para></para>
337     /// </param>
338     [MethodImpl(MethodImplOptions.AggressiveInlining)]
339     protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
340         ↪ Bit<TLink>.PartialWrite(storedValue, size, 5, -5);
341     /// <summary>
342     /// <para>
343     /// Determines whether this instance get left is child value.
344     /// </para>
345     /// <para></para>
346     /// </summary>
347     /// <param name="value">
348     /// <para>The value.</para>
349     /// <para></para>
350     /// </param>
351     /// <returns>
352     /// <para>The bool</para>
353     /// <para></para>
354     /// </returns>
355     [MethodImpl(MethodImplOptions.AggressiveInlining)]
356     protected virtual bool GetLeftIsChildValue(TLink value)
357     {
358         unchecked
359         {
360             return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 4, 1));
361             //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 4, 1), default);
362         }
363     }
364     /// <summary>
365     /// <para>
366     /// Sets the left is child value using the specified stored value.
367     /// </para>

```

```

368    /// </para>
369    /// <para></para>
370    /// </summary>
371    /// <param name="storedValue">
372    /// <para>The stored value.</para>
373    /// <para></para>
374    /// </param>
375    /// <param name="value">
376    /// <para>The value.</para>
377    /// <para></para>
378    /// </param>
379    [MethodImpl(MethodImplOptions.AggressiveInlining)]
380    protected virtual void SetLeftIsChildValue(ref TLink storedValue, bool value)
381    {
382        unchecked
383        {
384            var previousValue = storedValue;
385            var modified = Bit<TLink>.PartialWrite(previousValue,
386                ↪ _boolToAddressConverter.Convert(value), 4, 1);
387            storedValue = modified;
388        }
389    }
390    /// <summary>
391    /// <para>
392    /// Determines whether this instance get right is child value.
393    /// </para>
394    /// <para></para>
395    /// </summary>
396    /// <param name="value">
397    /// <para>The value.</para>
398    /// <para></para>
399    /// </param>
400    /// <returns>
401    /// <para>The bool</para>
402    /// <para></para>
403    /// </returns>
404    [MethodImpl(MethodImplOptions.AggressiveInlining)]
405    protected virtual bool GetRightIsChildValue(TLink value)
406    {
407        unchecked
408        {
409            return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 3, 1));
410            //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 3, 1), default);
411        }
412    }
413    /// <summary>
414    /// <para>
415    /// Sets the right is child value using the specified stored value.
416    /// </para>
417    /// <para></para>
418    /// </summary>
419    /// <param name="storedValue">
420    /// <para>The stored value.</para>
421    /// <para></para>
422    /// </param>
423    /// <param name="value">
424    /// <para>The value.</para>
425    /// <para></para>
426    /// </param>
427    [MethodImpl(MethodImplOptions.AggressiveInlining)]
428    protected virtual void SetRightIsChildValue(ref TLink storedValue, bool value)
429    {
430        unchecked
431        {
432            var previousValue = storedValue;
433            var modified = Bit<TLink>.PartialWrite(previousValue,
434                ↪ _boolToAddressConverter.Convert(value), 3, 1);
435            storedValue = modified;
436        }
437    }
438    /// <summary>
439    /// <para>
440    /// Determines whether this instance is child.
441    /// </para>
442    /// <para></para>
443    /// </summary>

```

```

444     /// </summary>
445     /// <param name="parent">
446     /// <para>The parent.</para>
447     /// <para></para>
448     /// </param>
449     /// <param name="possibleChild">
450     /// <para>The possible child.</para>
451     /// <para></para>
452     /// </param>
453     /// <returns>
454     /// <para>The bool</para>
455     /// <para></para>
456     /// </returns>
457     [MethodImpl(MethodImplOptions.AggressiveInlining)]
458     protected bool IsChild(TLink parent, TLink possibleChild)
459     {
460         var parentSize = GetSize(parent);
461         var childSize = GetSizeOrZero(possibleChild);
462         return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
463     }
464
465     /// <summary>
466     /// <para>
467     /// Gets the balance value using the specified stored value.
468     /// </para>
469     /// <para></para>
470     /// </summary>
471     /// <param name="storedValue">
472     /// <para>The stored value.</para>
473     /// <para></para>
474     /// </param>
475     /// <returns>
476     /// <para>The sbyte</para>
477     /// <para></para>
478     /// </returns>
479     [MethodImpl(MethodImplOptions.AggressiveInlining)]
480     protected virtual sbyte GetBalanceValue(TLink storedValue)
481     {
482         unchecked
483         {
484             var value = _addressToInt32Converter.Convert(Bit<TLink>.PartialRead(storedValue,
485                 ↪ 0, 3));
486             value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
487                 ↪ end of sbyte
488             return (sbyte)value;
489         }
490     }
491
492     /// <summary>
493     /// <para>
494     /// Sets the balance value using the specified stored value.
495     /// </para>
496     /// <para></para>
497     /// </summary>
498     /// <param name="storedValue">
499     /// <para>The stored value.</para>
500     /// <para></para>
501     /// </param>
502     /// <param name="value">
503     /// <para>The value.</para>
504     /// <para></para>
505     /// </param>
506     [MethodImpl(MethodImplOptions.AggressiveInlining)]
507     protected virtual void SetBalanceValue(ref TLink storedValue, sbyte value)
508     {
509         unchecked
510         {
511             var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
512                 ↪ value & 3);
513             var modified = Bit<TLink>.PartialWrite(storedValue, packagedValue, 0, 3);
514             storedValue = modified;
515         }
516     }
517
518     /// <summary>
519     /// <para>
520     /// The zero.
521     /// </para>

```

```

519 /// <para></para>
520 /// </summary>
521 public TLink this[TLink index]
522 {
523     [MethodImpl(MethodImplOptions.AggressiveInlining)]
524     get
525     {
526         var root = GetTreeRoot();
527         if (GreaterOrEqualThan(index, GetSize(root)))
528         {
529             return Zero;
530         }
531         while (!EqualToZero(root))
532         {
533             var left = GetLeftOrDefault(root);
534             var leftSize = GetSizeOrZero(left);
535             if (LessThan(index, leftSize))
536             {
537                 root = left;
538                 continue;
539             }
540             if (AreEqual(index, leftSize))
541             {
542                 return root;
543             }
544             root = GetRightOrDefault(root);
545             index = Subtract(index, Increment(leftSize));
546         }
547         return Zero; // TODO: Impossible situation exception (only if tree structure
548                     ↪ broken)
549     }
550 }
551
552 /// <summary>
553 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
554 /// ↪ (концом).
555 /// </summary>
556 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
557 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
558 /// <returns>Индекс искомой связи.</returns>
559 [MethodImpl(MethodImplOptions.AggressiveInlining)]
560 public TLink Search(TLink source, TLink target)
561 {
562     var root = GetTreeRoot();
563     while (!EqualToZero(root))
564     {
565         ref var rootLink = ref GetLinkReference(root);
566         var rootSource = rootLink.Source;
567         var rootTarget = rootLink.Target;
568         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
569             ↪ node.Key < root.Key
570         {
571             root = GetLeftOrDefault(root);
572         }
573         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
574             ↪ node.Key > root.Key
575         {
576             root = GetRightOrDefault(root);
577         }
578         else // node.Key == root.Key
579         {
580             return root;
581         }
582     }
583     return Zero;
584 }
585
586 // TODO: Return indices range instead of references count
587 /// <summary>
588 /// <para>
589 /// Counts the usages using the specified link.
590 /// </para>
591 /// <para></para>
592 /// </summary>
593 /// <param name="link">
594 /// <para>The link.</para>
595 /// <para></para>
596 /// </param>

```

```

593     /// <returns>
594     /// <para>The link</para>
595     /// <para></para>
596     /// </returns>
597     [MethodImpl(MethodImplOptions.AggressiveInlining)]
598     public TLink CountUsages(TLink link)
599     {
600         var root = GetTreeRoot();
601         var total = GetSize(root);
602         var totalRightIgnore = Zero;
603         while (!EqualToZero(root))
604         {
605             var @base = GetBasePartValue(root);
606             if (LessOrEqualThan(@base, link))
607             {
608                 root = GetRightOrDefault(root);
609             }
610             else
611             {
612                 totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
613                 root = GetLeftOrDefault(root);
614             }
615         }
616         root = GetTreeRoot();
617         var totalLeftIgnore = Zero;
618         while (!EqualToZero(root))
619         {
620             var @base = GetBasePartValue(root);
621             if (GreaterOrEqualThan(@base, link))
622             {
623                 root = GetLeftOrDefault(root);
624             }
625             else
626             {
627                 totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
628                 root = GetRightOrDefault(root);
629             }
630         }
631         return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
632     }
633 }
634
635     /// <summary>
636     /// <para>
637     /// Eaches the usage using the specified link.
638     /// </para>
639     /// <para></para>
640     /// </summary>
641     /// <param name="link">
642     /// <para>The link.</para>
643     /// <para></para>
644     /// </param>
645     /// <param name="handler">
646     /// <para>The handler.</para>
647     /// <para></para>
648     /// </param>
649     /// <returns>
650     /// <para>The continue.</para>
651     /// <para></para>
652     /// </returns>
653     [MethodImpl(MethodImplOptions.AggressiveInlining)]
654     public TLink EachUsage(TLink link, Func<IList<TLink>, TLink> handler)
655     {
656         var root = GetTreeRoot();
657         if (EqualToZero(root))
658         {
659             return Continue;
660         }
661         TLink first = Zero, current = root;
662         while (!EqualToZero(current))
663         {
664             var @base = GetBasePartValue(current);
665             if (GreaterOrEqualThan(@base, link))
666             {
667                 if (AreEqual(@base, link))
668                 {
669                     first = current;
670                 }

```

```

671         current = GetLeftOrDefault(current);
672     }
673     else
674     {
675         current = GetRightOrDefault(current);
676     }
677 }
678 if (!EqualToZero(first))
679 {
680     current = first;
681     while (true)
682     {
683         if (AreEqual(handler(GetLinkValues(current)), Break))
684         {
685             return Break;
686         }
687         current = GetNext(current);
688         if (EqualToZero(current) || !AreEqual(GetBasePartValue(current), link))
689         {
690             break;
691         }
692     }
693 }
694 return Continue;
695 }
696
697 /// <summary>
698 /// <para>
699 /// Prints the node value using the specified node.
700 /// </para>
701 /// <para></para>
702 /// </summary>
703 /// <param name="node">
704 /// <para>The node.</para>
705 /// <para></para>
706 /// </param>
707 /// <param name="sb">
708 /// <para>The sb.</para>
709 /// <para></para>
710 /// </param>
711 [MethodImpl(MethodImplOptions.AggressiveInlining)]
712 protected override void PrintNodeValue(TLink node, StringBuilder sb)
713 {
714     ref var link = ref GetLinkReference(node);
715     sb.Append(' ');
716     sb.Append(link.Source);
717     sb.Append('-');
718     sb.Append('>');
719     sb.Append(link.Target);
720 }
721 }
722 }

```

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         /// <summary>
25         /// <para>

```

```

25     /// The default.
26     /// </para>
27     /// <para></para>
28     /// </summary>
29     private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
    ↪     UncheckedConverter<TLink, long>.Default;

30
31     /// <summary>
32     /// <para>
33     /// The break.
34     /// </para>
35     /// <para></para>
36     /// </summary>
37     protected readonly TLink Break;
38     /// <summary>
39     /// <para>
40     /// The continue.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     protected readonly TLink Continue;
45     /// <summary>
46     /// <para>
47     /// The links.
48     /// </para>
49     /// <para></para>
50     /// </summary>
51     protected readonly byte* Links;
52     /// <summary>
53     /// <para>
54     /// The header.
55     /// </para>
56     /// <para></para>
57     /// </summary>
58     protected readonly byte* Header;
59
60     /// <summary>
61     /// <para>
62     /// Initializes a new <see cref="LinksRecursionlessSizeBalancedTreeMethodsBase"/>
    ↪     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,
    ↪     byte* links, byte* header)
80     {
81         Links = links;
82         Header = header;
83         Break = constants.Break;
84         Continue = constants.Continue;
85     }
86
87     /// <summary>
88     /// <para>
89     /// Gets the tree root.
90     /// </para>
91     /// <para></para>
92     /// </summary>
93     /// <returns>
94     /// <para>The link</para>
95     /// <para></para>
96     /// </returns>
97     [MethodImpl(MethodImplOptions.AggressiveInlining)]
98     protected abstract TLink GetTreeRoot();
99
100    /// <summary>

```

```

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

```



```

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

```

```

250 }
251
252 /// <summary>
253 /// <para>
254 /// Determines whether this instance first is to the right of second.
255 /// </para>
256 /// <para></para>
257 /// </summary>
258 /// <param name="first">
259 /// <para>The first.</para>
260 /// <para></para>
261 /// </param>
262 /// <param name="second">
263 /// <para>The second.</para>
264 /// <para></para>
265 /// </param>
266 /// <returns>
267 /// <para>The bool</para>
268 /// <para></para>
269 /// </returns>
270 [MethodImpl(MethodImplOptions.AggressiveInlining)]
271 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
272 {
273     ref var firstLink = ref GetLinkReference(first);
274     ref var secondLink = ref GetLinkReference(second);
275     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
276         ↪ secondLink.Source, secondLink.Target);
277 }
278
279 /// <summary>
280 /// <para>
281 /// The zero.
282 /// </para>
283 /// <para></para>
284 /// </summary>
285 public TLink this[TLink index]
286 {
287     [MethodImpl(MethodImplOptions.AggressiveInlining)]
288     get
289     {
290         var root = GetTreeRoot();
291         if (GreaterOrEqualThan(index, GetSize(root)))
292         {
293             return Zero;
294         }
295         while (!EqualToZero(root))
296         {
297             var left = GetLeftOrDefault(root);
298             var leftSize = GetSizeOrZero(left);
299             if (LessThan(index, leftSize))
300             {
301                 root = left;
302                 continue;
303             }
304             if (AreEqual(index, leftSize))
305             {
306                 return root;
307             }
308             root = GetRightOrDefault(root);
309             index = Subtract(index, Increment(leftSize));
310         }
311         return Zero; // TODO: Impossible situation exception (only if tree structure
312             ↪ broken)
313     }
314 }
315
316 /// <summary>
317 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
318 ↪ (концом).
319 /// </summary>
320 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
321 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
322 /// <returns>Индекс искомой связи.</returns>
323 [MethodImpl(MethodImplOptions.AggressiveInlining)]
324 public TLink Search(TLink source, TLink target)
325 {
326     var root = GetTreeRoot();
327     while (!EqualToZero(root))

```

```

325     {
326         ref var rootLink = ref GetLinkReference(root);
327         var rootSource = rootLink.Source;
328         var rootTarget = rootLink.Target;
329         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
330             ↪ node.Key < root.Key
331         {
332             root = GetLeftOrDefault(root);
333         }
334         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
335             ↪ node.Key > root.Key
336         {
337             root = GetRightOrDefault(root);
338         }
339         else // node.Key == root.Key
340         {
341             return root;
342         }
343     }
344     return Zero;
345 }
346
347 // TODO: Return indices range instead of references count
348 /// <summary>
349 /// <para>
350 /// Counts the usages using the specified link.
351 /// </para>
352 /// <para></para>
353 /// </summary>
354 /// <param name="link">
355 /// <para>The link.</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 CountUsages(TLink link)
364 {
365     var root = GetTreeRoot();
366     var total = GetSize(root);
367     var totalRightIgnore = Zero;
368     while (!EqualToZero(root))
369     {
370         var @base = GetBasePartValue(root);
371         if (LessOrEqualThan(@base, link))
372         {
373             root = GetRightOrDefault(root);
374         }
375         else
376         {
377             totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
378             root = GetLeftOrDefault(root);
379         }
380     }
381     root = GetTreeRoot();
382     var totalLeftIgnore = Zero;
383     while (!EqualToZero(root))
384     {
385         var @base = GetBasePartValue(root);
386         if (GreaterOrEqualThan(@base, link))
387         {
388             root = GetLeftOrDefault(root);
389         }
390         else
391         {
392             totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
393             root = GetRightOrDefault(root);
394         }
395     }
396     return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
397 }
398
399 /// <summary>
400 /// <para>
401 /// Eaches the usage using the specified base.
402 /// </para>

```

```

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

```

```

477         return @break;
478     }
479 }
480 return @continue;
481 }
482
483 /// <summary>
484 /// <para>
485 /// Prints the node value using the specified node.
486 /// </para>
487 /// <para></para>
488 /// </summary>
489 /// <param name="node">
490 /// <para>The node.</para>
491 /// <para></para>
492 /// </param>
493 /// <param name="sb">
494 /// <para>The sb.</para>
495 /// <para></para>
496 /// </param>
497 [MethodImpl(MethodImplOptions.AggressiveInlining)]
498 protected override void PrintNodeValue(TLink node, StringBuilder sb)
499 {
500     ref var link = ref GetLinkReference(node);
501     sb.Append(' ');
502     sb.Append(link.Source);
503     sb.Append('-');
504     sb.Append('>');
505     sb.Append(link.Target);
506 }
507 }
508 }

```

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         /// <summary>
25         /// <para>
26         /// The default.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
31             ↳ UncheckedConverter<TLink, long>.Default;
32
33         /// <summary>
34         /// <para>
35         /// The break.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         protected readonly TLink Break;
40
41         /// <summary>
42         /// <para>
43         /// The continue.
44         /// </para>
45         /// <para></para>
46         /// </summary>

```

```

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

```

```

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

```

```

196    /// <para></para>
197    /// </param>
198    /// <returns>
199    /// <para>A ref raw link of t link</para>
200    /// <para></para>
201    /// </returns>
202    [MethodImpl(MethodImplOptions.AggressiveInlining)]
203    protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
        ↪ AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
        ↪ _addressToInt64Converter.Convert(link)));

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

```



```

271 protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
272 {
273     ref var firstLink = ref GetLinkReference(first);
274     ref var secondLink = ref GetLinkReference(second);
275     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
        ↪ secondLink.Source, secondLink.Target);
276 }
277
278 /// <summary>
279 /// <para>
280 /// The zero.
281 /// </para>
282 /// <para></para>
283 /// </summary>
284 public TLink this[TLink index]
285 {
286     [MethodImpl(MethodImplOptions.AggressiveInlining)]
287     get
288     {
289         var root = GetTreeRoot();
290         if (GreaterOrEqualThan(index, GetSize(root)))
291         {
292             return Zero;
293         }
294         while (!EqualToZero(root))
295         {
296             var left = GetLeftOrDefault(root);
297             var leftSize = GetSizeOrZero(left);
298             if (LessThan(index, leftSize))
299             {
300                 root = left;
301                 continue;
302             }
303             if (AreEqual(index, leftSize))
304             {
305                 return root;
306             }
307             root = GetRightOrDefault(root);
308             index = Subtract(index, Increment(leftSize));
309         }
310         return Zero; // TODO: Impossible situation exception (only if tree structure
        ↪ broken)
311     }
312 }
313
314 /// <summary>
315 /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
        ↪ (концом).
316 /// </summary>
317 /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
318 /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
319 /// <returns>Индекс искомой связи.</returns>
320 [MethodImpl(MethodImplOptions.AggressiveInlining)]
321 public TLink Search(TLink source, TLink target)
322 {
323     var root = GetTreeRoot();
324     while (!EqualToZero(root))
325     {
326         ref var rootLink = ref GetLinkReference(root);
327         var rootSource = rootLink.Source;
328         var rootTarget = rootLink.Target;
329         if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
            ↪ node.Key < root.Key
330         {
331             root = GetLeftOrDefault(root);
332         }
333         else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
            ↪ node.Key > root.Key
334         {
335             root = GetRightOrDefault(root);
336         }
337         else // node.Key == root.Key
338         {
339             return root;
340         }
341     }
342     return Zero;
343 }

```

```

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

```

```

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

```

```

498     protected override void PrintNodeValue(TLink node, StringBuilder sb)
499     {
500         ref var link = ref GetLinkReference(node);
501         sb.Append(' ');
502         sb.Append(link.Source);
503         sb.Append('-');
504         sb.Append('>');
505         sb.Append(link.Target);
506     }
507 }
508 }

```

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

```

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

```

```

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

```

```

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 link</para>
150     /// <para></para>
151     /// </returns>
152     [MethodImpl(MethodImplOptions.AggressiveInlining)]
153     protected override TLink GetSize(TLink node) =>
154     ↪ GetSizeValue(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) => SetSizeValue(ref
172     ↪ GetLinkReference(node).SizeAsSource, size);
173
174     /// <summary>
175     /// <para>
176     /// Determines whether this instance get left is child.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     /// <param name="node">
181     /// <para>The node.</para>
182     /// <para></para>
183     /// </param>
184     /// <returns>
185     /// <para>The bool</para>
186     /// <para></para>
187     /// </returns>
188     [MethodImpl(MethodImplOptions.AggressiveInlining)]
189     protected override bool GetLeftIsChild(TLink node) =>
190     ↪ GetLeftIsChildValue(GetLinkReference(node).SizeAsSource);
191
192     /// <summary>
193     /// <para>
194     /// Sets the left is child using the specified node.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="node">
199     /// <para>The node.</para>
200     /// <para></para>
201     /// </param>
202     /// <param name="value">
203     /// <para>The value.</para>
204     /// <para></para>
205     /// </param>
206     [MethodImpl(MethodImplOptions.AggressiveInlining)]
207     protected override void SetLeftIsChild(TLink node, bool value) =>
208     ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
209
210     /// <summary>
211     /// <para>
212     /// Determines whether this instance get right is child.
213     /// </para>
214     /// <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) =>
223        ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsSource);
224
225    /// <summary>
226    /// <para>
227    /// <para>Sets the right is child using the specified node.
228    /// </para>
229    /// <para></para>
230    /// </summary>
231    /// <param name="node">
232    /// <para>The node.</para>
233    /// <para></para>
234    /// </param>
235    /// <param name="value">
236    /// <para>The value.</para>
237    /// <para></para>
238    /// </param>
239    [MethodImpl(MethodImplOptions.AggressiveInlining)]
240    protected override void SetRightIsChild(TLink node, bool value) =>
241        ↪ SetRightIsChildValue(ref GetLinkReference(node).SizeAsSource, value);
242
243    /// <summary>
244    /// <para>
245    /// <para>Gets the balance using the specified node.
246    /// </para>
247    /// <para></para>
248    /// </summary>
249    /// <param name="node">
250    /// <para>The node.</para>
251    /// <para></para>
252    /// </param>
253    /// <returns>
254    /// <para>The sbyte</para>
255    /// <para></para>
256    /// </returns>
257    [MethodImpl(MethodImplOptions.AggressiveInlining)]
258    protected override sbyte GetBalance(TLink node) =>
259        ↪ GetBalanceValue(GetLinkReference(node).SizeAsSource);
260
261    /// <summary>
262    /// <para>
263    /// <para>Sets the balance using the specified node.
264    /// </para>
265    /// <para></para>
266    /// </summary>
267    /// <param name="node">
268    /// <para>The node.</para>
269    /// <para></para>
270    /// </param>
271    /// <param name="value">
272    /// <para>The value.</para>
273    /// <para></para>
274    /// </param>
275    [MethodImpl(MethodImplOptions.AggressiveInlining)]
276    protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
277        ↪ GetLinkReference(node).SizeAsSource, value);
278
279    /// <summary>
280    /// <para>
281    /// <para>Gets the tree root.
282    /// </para>
283    /// <para></para>
284    /// </summary>
285    /// <returns>
286    /// <para>The link</para>
287    /// <para></para>
288    /// </returns>
289    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

286     protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
287
288     /// <summary>
289     /// <para>
290     /// Gets the base part value using the specified link.
291     /// </para>
292     /// <para></para>
293     /// </summary>
294     /// <param name="link">
295     /// <para>The link.</para>
296     /// <para></para>
297     /// </param>
298     /// <returns>
299     /// <para>The link</para>
300     /// <para></para>
301     /// </returns>
302     [MethodImpl(MethodImplOptions.AggressiveInlining)]
303     protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
304
305     /// <summary>
306     /// <para>
307     /// Determines whether this instance first is to the left of second.
308     /// </para>
309     /// <para></para>
310     /// </summary>
311     /// <param name="firstSource">
312     /// <para>The first source.</para>
313     /// <para></para>
314     /// </param>
315     /// <param name="firstTarget">
316     /// <para>The first target.</para>
317     /// <para></para>
318     /// </param>
319     /// <param name="secondSource">
320     /// <para>The second source.</para>
321     /// <para></para>
322     /// </param>
323     /// <param name="secondTarget">
324     /// <para>The second target.</para>
325     /// <para></para>
326     /// </param>
327     /// <returns>
328     /// <para>The bool</para>
329     /// <para></para>
330     /// </returns>
331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
332     protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
333     ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
334     ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
335
336     /// <summary>
337     /// <para>
338     /// Determines whether this instance first is to the right of second.
339     /// </para>
340     /// <para></para>
341     /// </summary>
342     /// <param name="firstSource">
343     /// <para>The first source.</para>
344     /// <para></para>
345     /// </param>
346     /// <param name="firstTarget">
347     /// <para>The first target.</para>
348     /// <para></para>
349     /// </param>
350     /// <param name="secondSource">
351     /// <para>The second source.</para>
352     /// <para></para>
353     /// </param>
354     /// <param name="secondTarget">
355     /// <para>The second target.</para>
356     /// <para></para>
357     /// </param>
358     /// <returns>
359     /// <para>The bool</para>
360     /// <para></para>
361     /// </returns>
362     [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

361     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
362         ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
363         ↪ (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));
364
365     /// <summary>
366     /// <para>
367     /// Clears the node using the specified node.
368     /// </para>
369     /// <para></para>
370     /// </summary>
371     /// <param name="node">
372     /// <para>The node.</para>
373     /// <para></para>
374     /// </param>
375     [MethodImpl(MethodImplOptions.AggressiveInlining)]
376     protected override void ClearNode(TLink node)
377     {
378         ref var link = ref GetLinkReference(node);
379         link.LeftAsSource = Zero;
380         link.RightAsSource = Zero;
381         link.SizeAsSource = Zero;
382     }
383 }

```

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

```

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

```

```

50     /// </returns>
51     [MethodImpl(MethodImplOptions.AggressiveInlining)]
52     protected override ref TLink GetLeftReference(TLink node) => ref
53         ↪ GetLinkReference(node).LeftAsSource;
54
55     /// <summary>
56     /// <para>
57     /// Gets the right reference using the specified node.
58     /// </para>
59     /// </summary>
60     /// <param name="node">
61     /// <para>The node.</para>
62     /// </para>
63     /// </param>
64     /// <returns>
65     /// <para>The ref link</para>
66     /// </para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ref TLink GetRightReference(TLink node) => ref
70         ↪ GetLinkReference(node).RightAsSource;
71
72     /// <summary>
73     /// <para>
74     /// Gets the left using the specified node.
75     /// </para>
76     /// </summary>
77     /// <param name="node">
78     /// <para>The node.</para>
79     /// </para>
80     /// </param>
81     /// <returns>
82     /// <para>The link</para>
83     /// </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     /// </summary>
93     /// <param name="node">
94     /// <para>The node.</para>
95     /// </para>
96     /// </param>
97     /// <returns>
98     /// <para>The link</para>
99     /// </para>
100    /// </returns>
101
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    /// </summary>
110    /// <param name="node">
111    /// <para>The node.</para>
112    /// </para>
113    /// </param>
114    /// <param name="left">
115    /// <para>The left.</para>
116    /// </para>
117    /// </param>
118
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.

```

```

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

```

```

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

```

```

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

```

```

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

```

```

219     /// <para></para>
220     /// </param>
221     /// <param name="secondTarget">
222     /// <para>The second target.</para>
223     /// <para></para>
224     /// </param>
225     /// <returns>
226     /// <para>The bool</para>
227     /// <para></para>
228     /// </returns>
229     [MethodImpl(MethodImplOptions.AggressiveInlining)]
230     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
        ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
        ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));

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

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

```

1.83 ./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
    ↳ GetLinkReference(node).LeftAsTarget;
53
54      /// <summary>
55      /// <para>
56      /// Gets the right reference using the specified node.
57      /// </para>
58      /// <para></para>
59      /// </summary>
60      /// <param name="node">
61      /// <para>The node.</para>
62      /// <para></para>
63      /// </param>
64      /// <returns>
65      /// <para>The ref link</para>
66      /// <para></para>
67      /// </returns>
68      [MethodImpl(MethodImplOptions.AggressiveInlining)]
69      protected override ref TLink GetRightReference(TLink node) => ref
    ↳ GetLinkReference(node).RightAsTarget;
70
71      /// <summary>
72      /// <para>
73      /// Gets the left using the specified node.
74      /// </para>
75      /// <para></para>
76      /// </summary>
77      /// <param name="node">
78      /// <para>The node.</para>
79      /// <para></para>
80      /// </param>
81      /// <returns>
82      /// <para>The link</para>
83      /// <para></para>
84      /// </returns>

```

```

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

```

```

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

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

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(TLink node, bool value) =>
205    ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);

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(TLink node) =>
222    ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsTarget);

223    /// <summary>
224    /// <para>
225    /// Sets the right is child using the specified node.
226    /// </para>
227    /// <para></para>
228    /// </summary>
229    /// <param name="node">
230    /// <para>The node.</para>
231    /// <para></para>
232    /// </param>
233    /// </param>

```

```

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    /// <para></para>
247    /// </summary>
248    /// <param name="node">
249    /// <para>The node.</para>
250    /// <para></para>
251    /// </param>
252    /// <returns>
253    /// <para>The sbyte</para>
254    /// <para></para>
255    /// </returns>
256    [MethodImpl(MethodImplOptions.AggressiveInlining)]
257    protected override sbyte GetBalance(TLink node) =>
258        ↪ GetBalanceValue(GetLinkReference(node).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(TLink node, sbyte value) => SetBalanceValue(ref
276        ↪ GetLinkReference(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 link</para>
286    /// <para></para>
287    /// </returns>
288    [MethodImpl(MethodImplOptions.AggressiveInlining)]
289    protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
290
291    /// <summary>
292    /// <para>
293    /// Gets the base part value using the specified link.
294    /// </para>
295    /// <para></para>
296    /// </summary>
297    /// <param name="link">
298    /// <para>The link.</para>
299    /// <para></para>
300    /// </param>
301    /// <returns>
302    /// <para>The link</para>
303    /// <para></para>
304    /// </returns>
305    [MethodImpl(MethodImplOptions.AggressiveInlining)]
306    protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Target;
307
308    /// <summary>
309    /// <para>
310    /// Determines whether this instance first is to the left of second.
311    /// </para>

```

```

309     /// <para></para>
310     /// </summary>
311     /// <param name="firstSource">
312     /// <para>The first source.</para>
313     /// <para></para>
314     /// </param>
315     /// <param name="firstTarget">
316     /// <para>The first target.</para>
317     /// <para></para>
318     /// </param>
319     /// <param name="secondSource">
320     /// <para>The second source.</para>
321     /// <para></para>
322     /// </param>
323     /// <param name="secondTarget">
324     /// <para>The second target.</para>
325     /// <para></para>
326     /// </param>
327     /// <returns>
328     /// <para>The bool</para>
329     /// <para></para>
330     /// </returns>
331     [MethodImpl(MethodImplOptions.AggressiveInlining)]
332     protected override bool FirstIsToLeftOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => LessThan(firstTarget, secondTarget) ||
    ↪ (AreEqual(firstTarget, secondTarget) && LessThan(firstSource, secondSource));
333
334     /// <summary>
335     /// <para>
336     /// Determines whether this instance first is to the right of second.
337     /// </para>
338     /// <para></para>
339     /// </summary>
340     /// <param name="firstSource">
341     /// <para>The first source.</para>
342     /// <para></para>
343     /// </param>
344     /// <param name="firstTarget">
345     /// <para>The first target.</para>
346     /// <para></para>
347     /// </param>
348     /// <param name="secondSource">
349     /// <para>The second source.</para>
350     /// <para></para>
351     /// </param>
352     /// <param name="secondTarget">
353     /// <para>The second target.</para>
354     /// <para></para>
355     /// </param>
356     /// <returns>
357     /// <para>The bool</para>
358     /// <para></para>
359     /// </returns>
360     [MethodImpl(MethodImplOptions.AggressiveInlining)]
361     protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
    ↪ TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
    ↪ (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));
362
363     /// <summary>
364     /// <para>
365     /// Clears the node using the specified node.
366     /// </para>
367     /// <para></para>
368     /// </summary>
369     /// <param name="node">
370     /// <para>The node.</para>
371     /// <para></para>
372     /// </param>
373     [MethodImpl(MethodImplOptions.AggressiveInlining)]
374     protected override void ClearNode(TLink node)
375     {
376         ref var link = ref GetLinkReference(node);
377         link.LeftAsTarget = Zero;
378         link.RightAsTarget = Zero;
379         link.SizeAsTarget = Zero;
380     }
381 }
382 }

```

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

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

```

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

```

```

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

```



```

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

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

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

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

1.86 ./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         /// <summary>
21         /// <para>
22         /// The create source tree methods.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         private readonly Func<ILinksTreeMethods<TLink>> _createSourceTreeMethods;
27         /// <summary>
28         /// <para>
29         /// The create target tree methods.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         private readonly Func<ILinksTreeMethods<TLink>> _createTargetTreeMethods;
34         /// <summary>
35         /// <para>
36         /// The header.
37         /// </para>

```

```

38     /// <para></para>
39     /// </summary>
40     private byte* _header;
41     /// <summary>
42     /// <para>
43     /// The links.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     private byte* _links;
48
49     /// <summary>
50     /// <para>
51     /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
52     /// </para>
53     /// <para></para>
54     /// </summary>
55     /// <param name="address">
56     /// <para>A address.</para>
57     /// <para></para>
58     /// </param>
59     [MethodImpl(MethodImplOptions.AggressiveInlining)]
60     public UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
61
62     /// <summary>
63     /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
64     ↪ минимальным шагом расширения базы данных.
65     /// </summary>
66     /// <param name="address">Полный путь к файлу базы данных.</param>
67     /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
68     ↪ байтах.</param>
69     [MethodImpl(MethodImplOptions.AggressiveInlining)]
70     public UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
71     ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
72     ↪ memoryReservationStep) { }
73
74     /// <summary>
75     /// <para>
76     /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="memory">
81     /// <para>A memory.</para>
82     /// <para></para>
83     /// </param>
84     [MethodImpl(MethodImplOptions.AggressiveInlining)]
85     public UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
86     ↪ DefaultLinksSizeStep) { }
87
88     /// <summary>
89     /// <para>
90     /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="memory">
95     /// <para>A memory.</para>
96     /// <para></para>
97     /// </param>
98     /// <param name="memoryReservationStep">
99     /// <para>A memory reservation step.</para>
100    /// <para></para>
101    /// </param>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep) :
104    ↪ this(memory, memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
105    ↪ IndexTreeType.Default) { }
106
107    /// <summary>
108    /// <para>
109    /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
110    /// </para>
111    /// <para></para>
112    /// </summary>
113    /// <param name="memory">
114    /// <para>A memory.</para>
115    /// <para></para>
116    /// </param>

```

```

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

```

```

181     /// </returns>
182     [MethodImpl(MethodImplOptions.AggressiveInlining)]
183     protected override ref LinksHeader<TLink> GetHeaderReference() => ref
        ↳ AsRef<LinksHeader<TLink>>(_header);
184
185     /// <summary>
186     /// <para>
187     /// Gets the link reference using the specified link index.
188     /// </para>
189     /// <para></para>
190     /// </summary>
191     /// <param name="linkIndex">
192     /// <para>The link index.</para>
193     /// <para></para>
194     /// </param>
195     /// <returns>
196     /// <para>A ref raw link of t link</para>
197     /// <para></para>
198     /// </returns>
199     [MethodImpl(MethodImplOptions.AggressiveInlining)]
200     protected override ref RawLink<TLink> GetLinkReference(TLink linkIndex) => ref
        ↳ AsRef<RawLink<TLink>>(_links + (LinkSizeInBytes * ConvertToInt64(linkIndex)));
201 }
202 }

```

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          /// <summary>
26          /// <para>
27          /// The default.
28          /// </para>
29          /// <para></para>
30          /// </summary>
31          private static readonly EqualityComparer<TLink> _equalityComparer =
                ↳ EqualityComparer<TLink>.Default;
32          /// <summary>
33          /// <para>
34          /// The default.
35          /// </para>
36          /// <para></para>
37          /// </summary>
38          private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
39          /// <summary>
40          /// <para>
41          /// The default.
42          /// </para>
43          /// <para></para>
44          /// </summary>
45          private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
                ↳ UncheckedConverter<TLink, long>.Default;
46          /// <summary>
47          /// <para>
48          /// The default.
49          /// </para>
50          /// <para></para>
51          /// </summary>

```



```

52 private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
    ↳ UncheckedConverter<long, TLink>.Default;
53
54 /// <summary>
55 /// <para>
56 /// The zero.
57 /// </para>
58 /// <para></para>
59 /// </summary>
60 private static readonly TLink _zero = default;
61 /// <summary>
62 /// <para>
63 /// The zero.
64 /// </para>
65 /// <para></para>
66 /// </summary>
67 private static readonly TLink _one = Arithmetic.Increment(_zero);
68
69 /// <summary>Возвращает размер одной связи в байтах.</summary>
70 /// <remarks>
71 /// Используется только во вне класса, не рекомендуется использовать внутри.
72 /// Так как во вне не обязательно будет доступен unsafe C#.
73 /// </remarks>
74 public static readonly long LinkSizeInBytes = RawLink<TLink>.SizeInBytes;
75
76 /// <summary>
77 /// <para>
78 /// The size in bytes.
79 /// </para>
80 /// <para></para>
81 /// </summary>
82 public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
83
84 /// <summary>
85 /// <para>
86 /// The link size in bytes.
87 /// </para>
88 /// <para></para>
89 /// </summary>
90 public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
91
92 /// <summary>
93 /// <para>
94 /// The memory.
95 /// </para>
96 /// <para></para>
97 /// </summary>
98 protected readonly IResizableDirectMemory _memory;
99 /// <summary>
100 /// <para>
101 /// The memory reservation step.
102 /// </para>
103 /// <para></para>
104 /// </summary>
105 protected readonly long _memoryReservationStep;
106
107 /// <summary>
108 /// <para>
109 /// The targets tree methods.
110 /// </para>
111 /// <para></para>
112 /// </summary>
113 protected ILinksTreeMethods<TLink> TargetsTreeMethods;
114 /// <summary>
115 /// <para>
116 /// The sources tree methods.
117 /// </para>
118 /// <para></para>
119 /// </summary>
120 protected ILinksTreeMethods<TLink> SourcesTreeMethods;
121 // 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
122 /// <summary>
123 /// <para>
124 /// The unused links list methods.
125 /// </para>
126 /// <para></para>
127 /// </summary>

```

```

128     protected ILinksListMethods<TLink> UnusedLinksListMethods;
129
130     /// <summary>
131     /// Возвращает общее число связей находящихся в хранилище.
132     /// </summary>
133     protected virtual TLink Total
134     {
135         [MethodImpl(MethodImplOptions.AggressiveInlining)]
136         get
137         {
138             ref var header = ref GetHeaderReference();
139             return Subtract(header.AllocatedLinks, header.FreeLinks);
140         }
141     }
142
143     /// <summary>
144     /// <para>
145     /// Gets the constants value.
146     /// </para>
147     /// <para></para>
148     /// </summary>
149     public virtual LinksConstants<TLink> Constants
150     {
151         [MethodImpl(MethodImplOptions.AggressiveInlining)]
152         get;
153     }
154
155     /// <summary>
156     /// <para>
157     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
158     /// </para>
159     /// <para></para>
160     /// </summary>
161     /// <param name="memory">
162     /// <para>A memory.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="memoryReservationStep">
166     /// <para>A memory reservation step.</para>
167     /// <para></para>
168     /// </param>
169     /// <param name="constants">
170     /// <para>A constants.</para>
171     /// <para></para>
172     /// </param>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
175     ↪ memoryReservationStep, LinksConstants<TLink> constants)
176     {
177         _memory = memory;
178         _memoryReservationStep = memoryReservationStep;
179         Constants = constants;
180     }
181
182     /// <summary>
183     /// <para>
184     /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="memory">
189     /// <para>A memory.</para>
190     /// <para></para>
191     /// </param>
192     /// <param name="memoryReservationStep">
193     /// <para>A memory reservation step.</para>
194     /// <para></para>
195     /// </param>
196     [MethodImpl(MethodImplOptions.AggressiveInlining)]
197     protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
198     ↪ memoryReservationStep) : this(memory, memoryReservationStep,
199     ↪ Default<LinksConstants<TLink>>.Instance) { }
200
201     /// <summary>
202     /// <para>
203     /// Inits the memory.
204     /// </para>
205     /// <para></para>
206     /// </summary>

```

```

203 /// </summary>
204 /// <param name="memory">
205 /// <para>The memory.</para>
206 /// <para></para>
207 /// </param>
208 /// <param name="memoryReservationStep">
209 /// <para>The memory reservation step.</para>
210 /// <para></para>
211 /// </param>
212 [MethodImpl(MethodImplOptions.AggressiveInlining)]
213 protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
214 {
215     if (memory.ReservedCapacity < memoryReservationStep)
216     {
217         memory.ReservedCapacity = memoryReservationStep;
218     }
219     SetPointers(memory);
220     ref var header = ref GetHeaderReference();
221     // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
222     memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
223         ↪ LinkHeaderSizeInBytes;
224     // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
225     header.ReservedLinks = ConvertToAddress((memory.ReservedCapacity -
226         ↪ LinkHeaderSizeInBytes) / LinkSizeInBytes);
227 }
228
229 /// <summary>
230 /// <para>
231 /// Counts the restrictions.
232 /// </para>
233 /// <para></para>
234 /// </summary>
235 /// <param name="restrictions">
236 /// <para>The restrictions.</para>
237 /// <para></para>
238 /// </param>
239 /// <exception cref="NotSupportedException">
240 /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
241 /// <para></para>
242 /// </exception>
243 /// <returns>
244 /// <para>The link</para>
245 /// <para></para>
246 /// </returns>
247 [MethodImpl(MethodImplOptions.AggressiveInlining)]
248 public virtual TLink Count(IList<TLink> restrictions)
249 {
250     // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
251     if (restrictions.Count == 0)
252     {
253         return Total;
254     }
255     var constants = Constants;
256     var any = constants.Any;
257     var index = restrictions[constants.IndexPart];
258     if (restrictions.Count == 1)
259     {
260         if (AreEqual(index, any))
261         {
262             return Total;
263         }
264         return Exists(index) ? GetOne() : GetZero();
265     }
266     if (restrictions.Count == 2)
267     {
268         var value = restrictions[1];
269         if (AreEqual(index, any))
270         {
271             if (AreEqual(value, any))
272             {
273                 return Total; // Any - как отсутствие ограничения
274             }
275             return Add(SourcesTreeMethods.CountUsages(value),
276                 ↪ TargetsTreeMethods.CountUsages(value));
277         }
278         else
279         {
280             if (!Exists(index))
281             {

```

```

278         {
279             return GetZero();
280         }
281         if (AreEqual(value, any))
282         {
283             return GetOne();
284         }
285         ref var storedLinkValue = ref GetLinkReference(index);
286         if (AreEqual(storedLinkValue.Source, value) ||
287             ↪ AreEqual(storedLinkValue.Target, value))
288         {
289             return GetOne();
290         }
291         return GetZero();
292     }
293     if (restrictions.Count == 3)
294     {
295         var source = restrictions[constants.SourcePart];
296         var target = restrictions[constants.TargetPart];
297         if (AreEqual(index, any))
298         {
299             if (AreEqual(source, any) && AreEqual(target, any))
300             {
301                 return Total;
302             }
303             else if (AreEqual(source, any))
304             {
305                 return TargetsTreeMethods.CountUsages(target);
306             }
307             else if (AreEqual(target, any))
308             {
309                 return SourcesTreeMethods.CountUsages(source);
310             }
311             else //if(source != Any && target != Any)
312             {
313                 // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
314                 var link = SourcesTreeMethods.Search(source, target);
315                 return AreEqual(link, constants.Null) ? GetZero() : GetOne();
316             }
317         }
318         else
319         {
320             if (!Exists(index))
321             {
322                 return GetZero();
323             }
324             if (AreEqual(source, any) && AreEqual(target, any))
325             {
326                 return GetOne();
327             }
328             ref var storedLinkValue = ref GetLinkReference(index);
329             if (!AreEqual(source, any) && !AreEqual(target, any))
330             {
331                 if (AreEqual(storedLinkValue.Source, source) &&
332                     ↪ AreEqual(storedLinkValue.Target, target))
333                 {
334                     return GetOne();
335                 }
336                 return GetZero();
337             }
338             var value = default(TLink);
339             if (AreEqual(source, any))
340             {
341                 value = target;
342             }
343             if (AreEqual(target, any))
344             {
345                 value = source;
346             }
347             if (AreEqual(storedLinkValue.Source, value) ||
348                 ↪ AreEqual(storedLinkValue.Target, value))
349             {
350                 return GetOne();
351             }
352             return GetZero();
353         }
354     }
355 }

```

```

353         throw new NotSupportedException("Другие размеры и способы ограничений не
           ↳ поддерживаются.");
354     }
355
356     /// <summary>
357     /// <para>
358     /// Eaches the handler.
359     /// </para>
360     /// <para></para>
361     /// </summary>
362     /// <param name="handler">
363     /// <para>The handler.</para>
364     /// <para></para>
365     /// </param>
366     /// <param name="restrictions">
367     /// <para>The restrictions.</para>
368     /// <para></para>
369     /// </param>
370     /// <exception cref="NotSupportedException">
371     /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
372     /// <para></para>
373     /// </exception>
374     /// <returns>
375     /// <para>The link</para>
376     /// <para></para>
377     /// </returns>
378     [MethodImpl(MethodImplOptions.AggressiveInlining)]
379     public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
380     {
381         var constants = Constants;
382         var @break = constants.Break;
383         if (restrictions.Count == 0)
384         {
385             for (var link = GetOne(); LessOrEqualThan(link,
386                 ↳ GetHeaderReference().AllocatedLinks); link = Increment(link))
387             {
388                 if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
389                 {
390                     return @break;
391                 }
392             }
393             return @break;
394         }
395         var @continue = constants.Continue;
396         var any = constants.Any;
397         var index = restrictions[constants.IndexPart];
398         if (restrictions.Count == 1)
399         {
400             if (AreEqual(index, any))
401             {
402                 return Each(handler, Array.Empty<TLink>());
403             }
404             if (!Exists(index))
405             {
406                 return @continue;
407             }
408             return handler(GetLinkStruct(index));
409         }
410         if (restrictions.Count == 2)
411         {
412             var value = restrictions[1];
413             if (AreEqual(index, any))
414             {
415                 if (AreEqual(value, any))
416                 {
417                     return Each(handler, Array.Empty<TLink>());
418                 }
419                 if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
420                 {
421                     return @break;
422                 }
423                 return Each(handler, new Link<TLink>(index, any, value));
424             }
425             else
426             {
427                 if (!Exists(index))
428                 {
429                     return @continue;
430                 }
431             }
432         }
433     }

```

```

429     }
430     if (AreEqual(value, any))
431     {
432         return handler(GetLinkStruct(index));
433     }
434     ref var storedLinkValue = ref GetLinkReference(index);
435     if (AreEqual(storedLinkValue.Source, value) ||
436         AreEqual(storedLinkValue.Target, value))
437     {
438         return handler(GetLinkStruct(index));
439     }
440     return @continue;
441 }
442 }
443 if (restrictions.Count == 3)
444 {
445     var source = restrictions[constants.SourcePart];
446     var target = restrictions[constants.TargetPart];
447     if (AreEqual(index, any))
448     {
449         if (AreEqual(source, any) && AreEqual(target, any))
450         {
451             return Each(handler, Array.Empty<TLink>());
452         }
453         else if (AreEqual(source, any))
454         {
455             return TargetsTreeMethods.EachUsage(target, handler);
456         }
457         else if (AreEqual(target, any))
458         {
459             return SourcesTreeMethods.EachUsage(source, handler);
460         }
461         else //if(source != Any && target != Any)
462         {
463             var link = SourcesTreeMethods.Search(source, target);
464             return AreEqual(link, constants.Null) ? @continue :
465                 ↪ handler(GetLinkStruct(link));
466         }
467     }
468     else
469     {
470         if (!Exists(index))
471         {
472             return @continue;
473         }
474         if (AreEqual(source, any) && AreEqual(target, any))
475         {
476             return handler(GetLinkStruct(index));
477         }
478         ref var storedLinkValue = ref GetLinkReference(index);
479         if (!AreEqual(source, any) && !AreEqual(target, any))
480         {
481             if (AreEqual(storedLinkValue.Source, source) &&
482                 AreEqual(storedLinkValue.Target, target))
483             {
484                 return handler(GetLinkStruct(index));
485             }
486             return @continue;
487         }
488         var value = default(TLink);
489         if (AreEqual(source, any))
490         {
491             value = target;
492         }
493         if (AreEqual(target, any))
494         {
495             value = source;
496         }
497         if (AreEqual(storedLinkValue.Source, value) ||
498             AreEqual(storedLinkValue.Target, value))
499         {
500             return handler(GetLinkStruct(index));
501         }
502         return @continue;
503     }
504 }
505 throw new NotSupportedException("Другие размеры и способы ограничений не
506     ↪ поддерживаются.");

```

```

505 }
506
507 /// <remarks>
508 /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
509   ↳ в другом месте (но не в менеджере памяти, а в логике Links)
510 /// </remarks>
511 [MethodImpl(MethodImplOptions.AggressiveInlining)]
512 public virtual TLink Update(ICollection<TLink> restrictions, ICollection<TLink> substitution)
513 {
514     var constants = Constants;
515     var @null = constants.Null;
516     var linkIndex = restrictions[constants.IndexPart];
517     ref var link = ref GetLinkReference(linkIndex);
518     ref var header = ref GetHeaderReference();
519     ref var firstAsSource = ref header.RootAsSource;
520     ref var firstAsTarget = ref header.RootAsTarget;
521     // Будет корректно работать только в том случае, если пространство выделенной связи
522     ↳ предварительно заполнено нулями
523     if (!AreEqual(link.Source, @null))
524     {
525         SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
526     }
527     if (!AreEqual(link.Target, @null))
528     {
529         TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
530     }
531     link.Source = substitution[constants.SourcePart];
532     link.Target = substitution[constants.TargetPart];
533     if (!AreEqual(link.Source, @null))
534     {
535         SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
536     }
537     if (!AreEqual(link.Target, @null))
538     {
539         TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
540     }
541     return linkIndex;
542 }
543
544 /// <remarks>
545 /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
546   ↳ пространство
547 /// </remarks>
548 [MethodImpl(MethodImplOptions.AggressiveInlining)]
549 public virtual TLink Create(ICollection<TLink> restrictions)
550 {
551     ref var header = ref GetHeaderReference();
552     var freeLink = header.FirstFreeLink;
553     if (!AreEqual(freeLink, Constants.Null))
554     {
555         UnusedLinksListMethods.Detach(freeLink);
556     }
557     else
558     {
559         {
560             var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
561             if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
562             {
563                 throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
564             }
565             if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
566             {
567                 _memory.ReservedCapacity += _memory.ReservationStep;
568                 SetPointers(_memory);
569                 header = ref GetHeaderReference();
570                 header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
571                 ↳ LinkSizeInBytes);
572             }
573             freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
574             _memory.UsedCapacity += LinkSizeInBytes;
575         }
576     }
577     return freeLink;
578 }
579
580 /// <summary>
581 /// <para>
582 /// Deletes the restrictions.
583 /// </para>
584 /// <para></para>
585 /// </summary>

```

```

580 /// <param name="restrictions">
581 /// <para>The restrictions.</para>
582 /// </para></param>
583 [MethodImpl(MethodImplOptions.AggressiveInlining)]
584 public virtual void Delete(ICollection<TLink> restrictions)
585 {
586     ref var header = ref GetHeaderReference();
587     var link = restrictions[Constants.IndexPart];
588     if (LessThan(link, header.AllocatedLinks))
589     {
590         UnusedLinksListMethods.AttachAsFirst(link);
591     }
592     else if (AreEqual(link, header.AllocatedLinks))
593     {
594         header.AllocatedLinks = Decrement(header.AllocatedLinks);
595         _memory.UsedCapacity -= LinkSizeInBytes;
596         // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
597         //   ↳ пока не дойдём до первой существующей связи
598         // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
599         while (GreaterThan(header.AllocatedLinks, GetZero()) &&
600             //   ↳ IsUnusedLink(header.AllocatedLinks))
601         {
602             UnusedLinksListMethods.Detach(header.AllocatedLinks);
603             header.AllocatedLinks = Decrement(header.AllocatedLinks);
604             _memory.UsedCapacity -= LinkSizeInBytes;
605         }
606     }
607 }
608 /// <summary>
609 /// <para>
610 /// Gets the link struct using the specified link index.
611 /// </para>
612 /// </summary>
613 /// <param name="linkIndex">
614 /// <para>The link index.</para>
615 /// </para></param>
616 /// <returns>
617 /// <para>A list of t link</para>
618 /// </returns>
619 [MethodImpl(MethodImplOptions.AggressiveInlining)]
620 public ICollection<TLink> GetLinkStruct(TLink linkIndex)
621 {
622     ref var link = ref GetLinkReference(linkIndex);
623     return new Link<TLink>(linkIndex, link.Source, link.Target);
624 }
625 /// <remarks>
626 /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
627 //   ↳ адрес реально поменялся
628 ///
629 /// Указатель this.links может быть в том же месте,
630 /// так как 0-я связь не используется и имеет такой же размер как Header,
631 /// поэтому header размещается в том же месте, что и 0-я связь
632 /// </remarks>
633 [MethodImpl(MethodImplOptions.AggressiveInlining)]
634 protected abstract void SetPointers(IResizableDirectMemory memory);
635
636 /// <summary>
637 /// <para>
638 /// Resets the pointers.
639 /// </para>
640 /// </summary>
641 [MethodImpl(MethodImplOptions.AggressiveInlining)]
642 protected virtual void ResetPointers()
643 {
644     SourcesTreeMethods = null;
645     TargetsTreeMethods = null;
646     UnusedLinksListMethods = null;
647 }
648
649 /// <summary>
650 /// <para>

```



```

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

```

```

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

```

```

808    /// </param>
809    /// <param name="second">
810    /// <para>The second.</para>
811    /// <para></para>
812    /// </param>
813    /// <returns>
814    /// <para>The bool</para>
815    /// <para></para>
816    /// </returns>
817    [MethodImpl(MethodImplOptions.AggressiveInlining)]
818    protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
819        ↪ _comparer.Compare(first, second) <= 0;
820
821    /// <summary>
822    /// <para>
823    /// Determines whether this instance greater than.
824    /// </para>
825    /// <para></para>
826    /// </summary>
827    /// <param name="first">
828    /// <para>The first.</para>
829    /// <para></para>
830    /// </param>
831    /// <param name="second">
832    /// <para>The second.</para>
833    /// <para></para>
834    /// </param>
835    /// <returns>
836    /// <para>The bool</para>
837    /// <para></para>
838    /// </returns>
839    [MethodImpl(MethodImplOptions.AggressiveInlining)]
840    protected virtual bool GreaterThan(TLink first, TLink second) =>
841        ↪ _comparer.Compare(first, second) > 0;
842
843    /// <summary>
844    /// <para>
845    /// Determines whether this instance greater or equal than.
846    /// </para>
847    /// <para></para>
848    /// </summary>
849    /// <param name="first">
850    /// <para>The first.</para>
851    /// <para></para>
852    /// </param>
853    /// <param name="second">
854    /// <para>The second.</para>
855    /// <para></para>
856    /// </param>
857    /// <returns>
858    /// <para>The bool</para>
859    /// <para></para>
860    /// </returns>
861    [MethodImpl(MethodImplOptions.AggressiveInlining)]
862    protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
863        ↪ _comparer.Compare(first, second) >= 0;
864
865    /// <summary>
866    /// <para>
867    /// Converts the to int 64 using the specified value.
868    /// </para>
869    /// <para></para>
870    /// </summary>
871    /// <param name="value">
872    /// <para>The value.</para>
873    /// <para></para>
874    /// </param>
875    /// <returns>
876    /// <para>The long</para>
877    /// <para></para>
878    /// </returns>
879    [MethodImpl(MethodImplOptions.AggressiveInlining)]
880    protected virtual long ConvertToInt64(TLink value) =>
881        ↪ _addressToInt64Converter.Convert(value);
882
883    /// <summary>
884    /// <para>
885    /// Converts the to address using the specified value.

```

```

882     /// </para>
883     /// <para></para>
884     /// </summary>
885     /// <param name="value">
886     /// <para>The value.</para>
887     /// <para></para>
888     /// </param>
889     /// <returns>
890     /// <para>The link</para>
891     /// <para></para>
892     /// </returns>
893     [MethodImpl(MethodImplOptions.AggressiveInlining)]
894     protected virtual TLink ConvertToAddress(long value) =>
895         ↪ _int64ToAddressConverter.Convert(value);
896
897     /// <summary>
898     /// <para>
899     /// Adds the first.
900     /// </para>
901     /// <para></para>
902     /// </summary>
903     /// <param name="first">
904     /// <para>The first.</para>
905     /// <para></para>
906     /// </param>
907     /// <param name="second">
908     /// <para>The second.</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 Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
917         ↪ second);
918
919     /// <summary>
920     /// <para>
921     /// Subtracts the first.
922     /// </para>
923     /// <para></para>
924     /// </summary>
925     /// <param name="first">
926     /// <para>The first.</para>
927     /// <para></para>
928     /// </param>
929     /// <param name="second">
930     /// <para>The second.</para>
931     /// <para></para>
932     /// </param>
933     /// <returns>
934     /// <para>The link</para>
935     /// <para></para>
936     /// </returns>
937     [MethodImpl(MethodImplOptions.AggressiveInlining)]
938     protected virtual TLink Subtract(TLink first, TLink second) =>
939         ↪ Arithmetic<TLink>.Subtract(first, second);
940
941     /// <summary>
942     /// <para>
943     /// Increments the link.
944     /// </para>
945     /// <para></para>
946     /// </summary>
947     /// <param name="link">
948     /// <para>The link.</para>
949     /// <para></para>
950     /// </param>
951     /// <returns>
952     /// <para>The link</para>
953     /// <para></para>
954     /// </returns>
955     [MethodImpl(MethodImplOptions.AggressiveInlining)]
956     protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);

```

```

957     /// Decrements the link.
958     /// </para>
959     /// <para></para>
960     /// </summary>
961     /// <param name="link">
962     /// <para>The link.</para>
963     /// <para></para>
964     /// </param>
965     /// <returns>
966     /// <para>The link</para>
967     /// <para></para>
968     /// </returns>
969     [MethodImpl(MethodImplOptions.AggressiveInlining)]
970     protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
971
972     #region Disposable
973
974     /// <summary>
975     /// <para>
976     /// Gets the allow multiple dispose calls value.
977     /// </para>
978     /// <para></para>
979     /// </summary>
980     protected override bool AllowMultipleDisposeCalls
981     {
982         [MethodImpl(MethodImplOptions.AggressiveInlining)]
983         get => true;
984     }
985
986     /// <summary>
987     /// <para>
988     /// Disposes the manual.
989     /// </para>
990     /// <para></para>
991     /// </summary>
992     /// <param name="manual">
993     /// <para>The manual.</para>
994     /// <para></para>
995     /// </param>
996     /// <param name="wasDisposed">
997     /// <para>The was disposed.</para>
998     /// <para></para>
999     /// </param>
1000     [MethodImpl(MethodImplOptions.AggressiveInlining)]
1001     protected override void Dispose(bool manual, bool wasDisposed)
1002     {
1003         if (!wasDisposed)
1004         {
1005             ResetPointers();
1006             _memory.DisposeIfPossible();
1007         }
1008     }
1009
1010     #endregion
1011 }
1012 }

```

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         /// <summary>

```

```

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

```

```

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

```

```

173     /// <param name="element">
174     /// <para>The element.</para>
175     /// </para>
176     /// </param>
177     [MethodImpl(MethodImplOptions.AggressiveInlining)]
178     protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
        ↪ element;
179
180     /// <summary>
181     /// <para>
182     /// Sets the last using the specified element.
183     /// </para>
184     /// </summary>
185     /// </summary>
186     /// <param name="element">
187     /// <para>The element.</para>
188     /// </para>
189     /// </param>
190     [MethodImpl(MethodImplOptions.AggressiveInlining)]
191     protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
        ↪ element;
192
193     /// <summary>
194     /// <para>
195     /// Sets the previous using the specified element.
196     /// </para>
197     /// </summary>
198     /// </summary>
199     /// <param name="element">
200     /// <para>The element.</para>
201     /// </para>
202     /// </param>
203     /// <param name="previous">
204     /// <para>The previous.</para>
205     /// </para>
206     /// </param>
207     [MethodImpl(MethodImplOptions.AggressiveInlining)]
208     protected override void SetPrevious(TLink element, TLink previous) =>
        ↪ GetLinkReference(element).Source = previous;
209
210     /// <summary>
211     /// <para>
212     /// Sets the next using the specified element.
213     /// </para>
214     /// </summary>
215     /// </summary>
216     /// <param name="element">
217     /// <para>The element.</para>
218     /// </para>
219     /// </param>
220     /// <param name="next">
221     /// <para>The next.</para>
222     /// </para>
223     /// </param>
224     [MethodImpl(MethodImplOptions.AggressiveInlining)]
225     protected override void SetNext(TLink element, TLink next) =>
        ↪ GetLinkReference(element).Target = next;
226
227     /// <summary>
228     /// <para>
229     /// Sets the size using the specified size.
230     /// </para>
231     /// </summary>
232     /// </summary>
233     /// <param name="size">
234     /// <para>The size.</para>
235     /// </para>
236     /// </param>
237     [MethodImpl(MethodImplOptions.AggressiveInlining)]
238     protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
239 }
240 }

```

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         /// <summary>
19         /// <para>
20         /// The default.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private static readonly EqualityComparer<TLink> _equalityComparer =
25             ↳ EqualityComparer<TLink>.Default;
26
27         /// <summary>
28         /// <para>
29         /// The size.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         public static readonly long SizeInBytes = Structure<RawLink<TLink>>.Size;
34
35         /// <summary>
36         /// <para>
37         /// The source.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         public TLink Source;
42         /// <summary>
43         /// <para>
44         /// The target.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         public TLink Target;
49         /// <summary>
50         /// <para>
51         /// The left as source.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         public TLink LeftAsSource;
56         /// <summary>
57         /// <para>
58         /// The right as source.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         public TLink RightAsSource;
63         /// <summary>
64         /// <para>
65         /// The size as source.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         public TLink SizeAsSource;
70         /// <summary>
71         /// <para>
72         /// The left as target.
73         /// </para>
74         /// <para></para>
75         /// </summary>
76         public TLink LeftAsTarget;
77         /// <summary>
78         /// <para>
79         /// The right as target.
80         /// </para>
81         /// <para></para>
82         /// </summary>
83         public TLink RightAsTarget;

```

```

83     /// <summary>
84     /// <para>
85     /// The size as target.
86     /// </para>
87     /// <para></para>
88     /// </summary>
89     public TLink SizeAsTarget;
90
91     /// <summary>
92     /// <para>
93     /// Determines whether this instance equals.
94     /// </para>
95     /// <para></para>
96     /// </summary>
97     /// <param name="obj">
98     /// <para>The obj.</para>
99     /// <para></para>
100    /// </param>
101    /// <returns>
102    /// <para>The bool</para>
103    /// <para></para>
104    /// </returns>
105    [MethodImpl(MethodImplOptions.AggressiveInlining)]
106    public override bool Equals(object obj) => obj is RawLink<TLink> link ? Equals(link) :
        ↳ false;
107
108    /// <summary>
109    /// <para>
110    /// Determines whether this instance equals.
111    /// </para>
112    /// <para></para>
113    /// </summary>
114    /// <param name="other">
115    /// <para>The other.</para>
116    /// <para></para>
117    /// </param>
118    /// <returns>
119    /// <para>The bool</para>
120    /// <para></para>
121    /// </returns>
122    [MethodImpl(MethodImplOptions.AggressiveInlining)]
123    public bool Equals(RawLink<TLink> other)
124        => _equalityComparer.Equals(Source, other.Source)
125        && _equalityComparer.Equals(Target, other.Target)
126        && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
127        && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
128        && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
129        && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
130        && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
131        && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);
132
133    /// <summary>
134    /// <para>
135    /// Gets the hash code.
136    /// </para>
137    /// <para></para>
138    /// </summary>
139    /// <returns>
140    /// <para>The int</para>
141    /// <para></para>
142    /// </returns>
143    [MethodImpl(MethodImplOptions.AggressiveInlining)]
144    public override int GetHashCode() => (Source, Target, LeftAsSource, RightAsSource,
        ↳ SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();
145
146    [MethodImpl(MethodImplOptions.AggressiveInlining)]
147    public static bool operator ==(RawLink<TLink> left, RawLink<TLink> right) =>
        ↳ left.Equals(right);
148
149    [MethodImpl(MethodImplOptions.AggressiveInlining)]
150    public static bool operator !=(RawLink<TLink> left, RawLink<TLink> right) => !(left ==
        ↳ right);
151 }
152 }

```

1.90 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethod

```

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        /// <summary>
26        /// <para>
27        /// The header.
28        /// </para>
29        /// <para></para>
30        /// </summary>
31        protected new readonly LinksHeader<uint>* Header;
32        /// <summary>
33        /// <para>
34        /// Initializes a new <see cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
35        ↪ instance.
36        /// </para>
37        /// <para></para>
38        /// </summary>
39        /// <param name="constants">
40        /// <para>A constants.</para>
41        /// <para></para>
42        /// </param>
43        /// <param name="links">
44        /// <para>A links.</para>
45        /// <para></para>
46        /// </param>
47        /// <param name="header">
48        /// <para>A header.</para>
49        /// <para></para>
50        /// </param>
51        protected UInt32LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<uint>
52        ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header)
53        : base(constants, (byte*)links, (byte*)header)
54        {
55            Links = links;
56            Header = header;
57        }
58        /// <summary>
59        /// <para>
60        /// Gets the zero.
61        /// </para>
62        /// <para></para>
63        /// </summary>
64        /// <returns>
65        /// <para>The uint</para>
66        /// <para></para>
67        /// </returns>
68        [MethodImpl(MethodImplOptions.AggressiveInlining)]
69        protected override uint GetZero() => 0U;
70        /// <summary>
71        /// <para>
72        /// Determines whether this instance equal to zero.
73        /// </para>
74        /// <para></para>
75        /// </summary>
76        /// <param name="value">
77        /// <para>The value.</para>

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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 }

```

1.91 ./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 :
        ↪ LinksSizeBalancedTreeMethodsBase<uint>
16    {
17        /// <summary>
18        /// <para>
19        /// The links.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        protected new readonly RawLink<uint>* Links;
24        /// <summary>
25        /// <para>
26        /// The header.
27        /// </para>
28        /// <para></para>
29        /// </summary>
30        protected new readonly LinksHeader<uint>* Header;
31
32        /// <summary>
33        /// <para>
34        /// Initializes a new <see cref="UInt32LinksSizeBalancedTreeMethodsBase"/> instance.
35        /// </para>
36        /// <para></para>
37        /// </summary>
38        /// <param name="constants">
39        /// <para>A constants.</para>
40        /// <para></para>
41        /// </param>

```



```

42     /// <param name="links">
43     /// <para>A links.</para>
44     /// </para>
45     /// </param>
46     /// <param name="header">
47     /// <para>A header.</para>
48     /// </para>
49     /// </param>
50     protected UInt32LinksSizeBalancedTreeMethodsBase(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     /// </summary>
62     /// <returns>
63     /// <para>The uint</para>
64     /// </returns>
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     protected override uint GetZero() => 0U;
67
68     /// <summary>
69     /// <para>
70     /// Determines whether this instance equal to zero.
71     /// </para>
72     /// </summary>
73     /// <param name="value">
74     /// <para>The value.</para>
75     /// </para>
76     /// </param>
77     /// <returns>
78     /// <para>The bool</para>
79     /// </returns>
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     protected override bool EqualToZero(uint value) => value == 0U;
82
83     /// <summary>
84     /// <para>
85     /// Determines whether this instance are equal.
86     /// </para>
87     /// </summary>
88     /// <param name="first">
89     /// <para>The first.</para>
90     /// </para>
91     /// </param>
92     /// <param name="second">
93     /// <para>The second.</para>
94     /// </para>
95     /// </param>
96     /// <returns>
97     /// <para>The bool</para>
98     /// </returns>
99     [MethodImpl(MethodImplOptions.AggressiveInlining)]
100    protected override bool AreEqual(uint first, uint second) => first == second;
101
102     /// <summary>
103     /// <para>
104     /// Determines whether this instance greater than zero.
105     /// </para>
106     /// </summary>
107     /// <param name="value">
108     /// <para>The value.</para>
109     /// </para>
110     /// </param>
111     /// <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 }

```

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

```

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

```

```

6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links sources recursionless size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods :
15        ↳ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see
20        ↳ cref="UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="constants">
25        /// <para>A constants.</para>
26        /// <para></para>
27        /// </param>
28        /// <param name="links">
29        /// <para>A links.</para>
30        /// <para></para>
31        /// </param>
32        /// <param name="header">
33        /// <para>A header.</para>
34        /// <para></para>
35        /// </param>
36        public UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
37        ↳ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
38        ↳ header) { }
39
40        /// <summary>
41        /// <para>
42        /// Gets the left reference using the specified node.
43        /// </para>
44        /// <para></para>
45        /// </summary>
46        /// <param name="node">
47        /// <para>The node.</para>
48        /// <para></para>
49        /// </param>
50        /// <returns>
51        /// <para>The ref uint</para>
52        /// <para></para>
53        /// </returns>
54        [MethodImpl(MethodImplOptions.AggressiveInlining)]
55        protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
56
57        /// <summary>
58        /// <para>
59        /// Gets the right reference using the specified node.
60        /// </para>
61        /// <para></para>
62        /// </summary>
63        /// <param name="node">
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].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 ref uint</para>
87        /// <para></para>
88        /// </returns>
89        [MethodImpl(MethodImplOptions.AggressiveInlining)]
90        protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;
91
92        /// <summary>
93        /// <para>
94        /// Gets the right using the specified node.
95        /// </para>
96        /// <para></para>
97        /// </summary>
98        /// <param name="node">
99        /// <para>The node.</para>
100       /// <para></para>
101       /// </param>
102       /// <returns>
103       /// <para>The ref uint</para>
104       /// <para></para>
105       /// </returns>
106       [MethodImpl(MethodImplOptions.AggressiveInlining)]
107       protected override ref uint GetRightReference(uint node) => ref Links[node].RightAsSource;
108   }
109 }

```

```

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 <
    ↪ secondTarget);

```



```

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

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

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links sources size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksSourcesSizeBalancedTreeMethods :
    ↪ UInt32LinksSizeBalancedTreeMethodsBase
15    {
16        /// <summary>
17        /// <para>
18        /// Initializes a new <see cref="UInt32LinksSourcesSizeBalancedTreeMethods"/> instance.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <param name="constants">
23        /// <para>A constants.</para>

```

```

24     /// <para></para>
25     /// </param>
26     /// <param name="links">
27     /// <para>A links.</para>
28     /// <para></para>
29     /// </param>
30     /// <param name="header">
31     /// <para>A header.</para>
32     /// <para></para>
33     /// </param>
34     public UInt32LinksSourcesSizeBalancedTreeMethods(LinksConstants<uint> constants,
35     ↪ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
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>

```

```

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

```

```

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

```

1.94 ./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 :
    ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see
    ↪ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
    ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
    ↪ header) { }
35
36         /// <summary>
37         /// <para>
38         /// Gets the left reference using the specified node.
39         /// </para>
40         /// <para></para>
41         /// </summary>

```

```

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

```

```

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

```

```

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

```



```

269     /// <para>The node.</para>
270     /// <para></para>
271     /// </param>
272     [MethodImpl(MethodImplOptions.AggressiveInlining)]
273     protected override void ClearNode(uint node)
274     {
275         ref var link = ref Links[node];
276         link.LeftAsTarget = 0U;
277         link.RightAsTarget = 0U;
278         link.SizeAsTarget = 0U;
279     }
280 }
281 }

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Memory.United.Specific
6 {
7     /// <summary>
8     /// <para>
9     /// Represents the int 32 links targets size balanced tree methods.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
14    public unsafe class UInt32LinksTargetsSizeBalancedTreeMethods :
15        ↳ UInt32LinksSizeBalancedTreeMethodsBase
16    {
17        /// <summary>
18        /// <para>
19        /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> instance.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="constants">
24        /// <para>A constants.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="links">
28        /// <para>A links.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="header">
32        /// <para>A header.</para>
33        /// <para></para>
34        /// </param>
35        public UInt32LinksTargetsSizeBalancedTreeMethods(LinksConstants<uint> constants,
36            ↳ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }
37
38        /// <summary>
39        /// <para>
40        /// Gets the left reference using the specified node.
41        /// </para>
42        /// <para></para>
43        /// </summary>
44        /// <param name="node">
45        /// <para>The node.</para>
46        /// <para></para>
47        /// </param>
48        /// <returns>
49        /// <para>The ref uint</para>
50        /// <para></para>
51        /// </returns>
52        [MethodImpl(MethodImplOptions.AggressiveInlining)]
53        protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsTarget;
54
55        /// <summary>
56        /// <para>
57        /// Gets the right reference using the specified node.
58        /// </para>
59        /// <para></para>
60        /// </summary>
61        /// <param name="node">
62        /// <para>The node.</para>
63        /// <para></para>
64        /// </param>
65        /// <returns>
66        /// <para>The ref uint</para>
67        /// <para></para>
68        /// </returns>
69    }
70 }

```

```

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

```

```

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

```

```

216     /// <param name="secondSource">
217     /// <para>The second source.</para>
218     /// <para></para>
219     /// </param>
220     /// <param name="secondTarget">
221     /// <para>The second target.</para>
222     /// <para></para>
223     /// </param>
224     /// <returns>
225     /// <para>The bool</para>
226     /// <para></para>
227     /// </returns>
228     [MethodImpl(MethodImplOptions.AggressiveInlining)]
229     protected override bool FirstIsToTheLeftOfSecond(uint firstSource, uint firstTarget,
230     ↪ uint secondSource, uint secondTarget)
231     => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
232     ↪ secondSource);
233
234     /// <summary>
235     /// <para>
236     /// Determines whether this instance first is to the right of second.
237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <param name="firstSource">
241     /// <para>The first source.</para>
242     /// <para></para>
243     /// </param>
244     /// <param name="firstTarget">
245     /// <para>The first target.</para>
246     /// <para></para>
247     /// </param>
248     /// <param name="secondSource">
249     /// <para>The second source.</para>
250     /// <para></para>
251     /// </param>
252     /// <param name="secondTarget">
253     /// <para>The second target.</para>
254     /// <para></para>
255     /// </param>
256     /// <returns>
257     /// <para>The bool</para>
258     /// <para></para>
259     /// </returns>
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
262     ↪ uint secondSource, uint secondTarget)
263     => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
264     ↪ secondSource);
265
266     /// <summary>
267     /// <para>
268     /// Clears the node using the specified node.
269     /// </para>
270     /// <para></para>
271     /// </summary>
272     /// <param name="node">
273     /// <para>The node.</para>
274     /// <para></para>
275     /// </param>
276     [MethodImpl(MethodImplOptions.AggressiveInlining)]
277     protected override void ClearNode(uint node)
278     {
279         ref var link = ref Links[node];
280         link.LeftAsTarget = 0U;
281         link.RightAsTarget = 0U;
282         link.SizeAsTarget = 0U;
283     }
284 }
285 }

```

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

```

1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Memory;
4 using Platform.Singletons;
5 using Platform.Data.Doublets.Memory.United.Generic;
6

```

```

7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.Data.Doublets.Memory.United.Specific
10 {
11     /// <summary>
12     /// <para>Represents a low-level implementation of direct access to resizable memory, for
13     /// <para>organizing the storage of links with addresses represented as <see cref="uint" />.</para>
14     /// <para>Представляет низкоуровневую реализацию прямого доступа к памяти с переменным
15     /// <para>размером, для организации хранения связей с адресами представленными в виде <see
16     /// <para>cref="uint"/>.</para>
17     /// </summary>
18     public unsafe class UInt32UnitedMemoryLinks : UnitedMemoryLinksBase<uint>
19     {
20         /// <summary>
21         /// <para>
22         /// <para>The create source tree methods.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         private readonly Func<ILinksTreeMethods<uint>> _createSourceTreeMethods;
27         /// <summary>
28         /// <para>
29         /// <para>The create target tree methods.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         private readonly Func<ILinksTreeMethods<uint>> _createTargetTreeMethods;
34         /// <summary>
35         /// <para>
36         /// <para>The header.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         private LinksHeader<uint>* _header;
41         /// <summary>
42         /// <para>
43         /// <para>The links.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         private RawLink<uint>* _links;
48
49         /// <summary>
50         /// <para>
51         /// <para>Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         /// <param name="address">
56         /// <para>A address.</para>
57         /// <para></para>
58         /// </param>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public UInt32UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
61
62         /// <summary>
63         /// <para>Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
64         /// <para>→ минимальным шагом расширения базы данных.
65         /// </summary>
66         /// <param name="address">Полный путь к файлу базы данных.</param>
67         /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
68         /// <para>→ байтах.</param>
69         [MethodImpl(MethodImplOptions.AggressiveInlining)]
70         public UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
71         /// <para>→ FileMappedResizableDirectMemory(address, memoryReservationStep),
72         /// <para>→ memoryReservationStep) { }
73
74         /// <summary>
75         /// <para>
76         /// <para>Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
77         /// </para>
78         /// <para></para>
79         /// </summary>
80         /// <param name="memory">
81         /// <para>A memory.</para>
82         /// <para></para>
83         /// </param>
84         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

78     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
    ↪     DefaultLinksSizeStep) { }
79
80     /// <summary>
81     /// <para>
82     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     /// <param name="memory">
87     /// <para>A memory.</para>
88     /// <para></para>
89     /// </param>
90     /// <param name="memoryReservationStep">
91     /// <para>A memory reservation step.</para>
92     /// <para></para>
93     /// </param>
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
    ↪     memoryReservationStep) : this(memory, memoryReservationStep,
    ↪     Default<LinksConstants<uint>>.Instance, IndexTreeType.Default) { }
96
97     /// <summary>
98     /// <para>
99     /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
100    /// </para>
101    /// <para></para>
102    /// </summary>
103    /// <param name="memory">
104    /// <para>A memory.</para>
105    /// <para></para>
106    /// </param>
107    /// <param name="memoryReservationStep">
108    /// <para>A memory reservation step.</para>
109    /// <para></para>
110    /// </param>
111    /// <param name="constants">
112    /// <para>A constants.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="indexTreeType">
116    /// <para>A index tree type.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    public UInt32UnitedMemoryLinks(IResizableDirectMemory memory, long
    ↪     memoryReservationStep, LinksConstants<uint> constants, IndexTreeType indexTreeType)
    ↪     : base(memory, memoryReservationStep, constants)
121    {
122        if (indexTreeType == IndexTreeType.SizeBalancedTree)
123        {
124            _createSourceTreeMethods = () => new
    ↪            UInt32LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
125            _createTargetTreeMethods = () => new
    ↪            UInt32LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
126        }
127        else
128        {
129            _createSourceTreeMethods = () => new
    ↪            UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
    ↪            _header);
130            _createTargetTreeMethods = () => new
    ↪            UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
    ↪            _header);
131        }
132        Init(memory, memoryReservationStep);
133    }
134
135    /// <summary>
136    /// <para>
137    /// Sets the pointers using the specified memory.
138    /// </para>
139    /// <para></para>
140    /// </summary>
141    /// <param name="memory">
142    /// <para>The memory.</para>
143    /// <para></para>

```

```

144 /// </param>
145 [MethodImpl(MethodImplOptions.AggressiveInlining)]
146 protected override void SetPointers(IResizableDirectMemory memory)
147 {
148     _header = (LinksHeader<uint>*)memory.Pointer;
149     _links = (RawLink<uint>*)memory.Pointer;
150     SourcesTreeMethods = _createSourceTreeMethods();
151     TargetsTreeMethods = _createTargetTreeMethods();
152     UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_links, _header);
153 }
154
155 /// <summary>
156 /// <para>
157 /// Resets the pointers.
158 /// </para>
159 /// <para></para>
160 /// </summary>
161 [MethodImpl(MethodImplOptions.AggressiveInlining)]
162 protected override void ResetPointers()
163 {
164     base.ResetPointers();
165     _links = null;
166     _header = null;
167 }
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 uint</para>
177 /// <para></para>
178 /// </returns>
179 [MethodImpl(MethodImplOptions.AggressiveInlining)]
180 protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
181
182 /// <summary>
183 /// <para>
184 /// Gets the link reference using the specified link index.
185 /// </para>
186 /// <para></para>
187 /// </summary>
188 /// <param name="linkIndex">
189 /// <para>The link index.</para>
190 /// <para></para>
191 /// </param>
192 /// <returns>
193 /// <para>A ref raw link of uint</para>
194 /// <para></para>
195 /// </returns>
196 [MethodImpl(MethodImplOptions.AggressiveInlining)]
197 protected override ref RawLink<uint> GetLinkReference(uint linkIndex) => ref
    ↪ _links[linkIndex];
198
199 /// <summary>
200 /// <para>
201 /// Determines whether this instance are equal.
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 AreEqual(uint first, uint second) => first == second;
219
220 /// <summary>

```

```

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

```



```

299     /// <para></para>
300     /// </returns>
301     [MethodImpl(MethodImplOptions.AggressiveInlining)]
302     protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;
303
304     /// <summary>
305     /// <para>
306     /// Gets the zero.
307     /// </para>
308     /// <para></para>
309     /// </summary>
310     /// <returns>
311     /// <para>The uint</para>
312     /// <para></para>
313     /// </returns>
314     [MethodImpl(MethodImplOptions.AggressiveInlining)]
315     protected override uint GetZero() => 0U;
316
317     /// <summary>
318     /// <para>
319     /// Gets the one.
320     /// </para>
321     /// <para></para>
322     /// </summary>
323     /// <returns>
324     /// <para>The uint</para>
325     /// <para></para>
326     /// </returns>
327     [MethodImpl(MethodImplOptions.AggressiveInlining)]
328     protected override uint GetOne() => 1U;
329
330     /// <summary>
331     /// <para>
332     /// Converts the to int 64 using the specified value.
333     /// </para>
334     /// <para></para>
335     /// </summary>
336     /// <param name="value">
337     /// <para>The value.</para>
338     /// <para></para>
339     /// </param>
340     /// <returns>
341     /// <para>The long</para>
342     /// <para></para>
343     /// </returns>
344     [MethodImpl(MethodImplOptions.AggressiveInlining)]
345     protected override long ConvertToInt64(uint value) => (long)value;
346
347     /// <summary>
348     /// <para>
349     /// Converts the to address using the specified value.
350     /// </para>
351     /// <para></para>
352     /// </summary>
353     /// <param name="value">
354     /// <para>The value.</para>
355     /// <para></para>
356     /// </param>
357     /// <returns>
358     /// <para>The uint</para>
359     /// <para></para>
360     /// </returns>
361     [MethodImpl(MethodImplOptions.AggressiveInlining)]
362     protected override uint ConvertToAddress(long value) => (uint)value;
363
364     /// <summary>
365     /// <para>
366     /// Adds the first.
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 uint</para>
380     /// <para></para>
381     /// </returns>
382     [MethodImpl(MethodImplOptions.AggressiveInlining)]
383     protected override uint Add(uint first, uint second) => first + second;
384
385     /// <summary>
386     /// <para>
387     /// Subtracts the first.
388     /// </para>
389     /// <para></para>
390     /// </summary>
391     /// <param name="first">
392     /// <para>The first.</para>
393     /// <para></para>
394     /// </param>
395     /// <param name="second">
396     /// <para>The second.</para>
397     /// <para></para>
398     /// </param>
399     /// <returns>
400     /// <para>The uint</para>
401     /// <para></para>
402     /// </returns>
403     [MethodImpl(MethodImplOptions.AggressiveInlining)]
404     protected override uint Subtract(uint first, uint second) => first - second;
405
406     /// <summary>
407     /// <para>
408     /// Increments the link.
409     /// </para>
410     /// <para></para>
411     /// </summary>
412     /// <param name="link">
413     /// <para>The link.</para>
414     /// <para></para>
415     /// </param>
416     /// <returns>
417     /// <para>The uint</para>
418     /// <para></para>
419     /// </returns>
420     [MethodImpl(MethodImplOptions.AggressiveInlining)]
421     protected override uint Increment(uint link) => ++link;
422
423     /// <summary>
424     /// <para>
425     /// Decrements the link.
426     /// </para>
427     /// <para></para>
428     /// </summary>
429     /// <param name="link">
430     /// <para>The link.</para>
431     /// <para></para>
432     /// </param>
433     /// <returns>
434     /// <para>The uint</para>
435     /// <para></para>
436     /// </returns>
437     [MethodImpl(MethodImplOptions.AggressiveInlining)]
438     protected override uint Decrement(uint link) => --link;
439 }
440 }

```

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         /// <summary>
18         /// <para>
19         /// The links.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         private readonly RawLink<uint>* _links;
24         /// <summary>
25         /// <para>
26         /// The header.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private readonly LinksHeader<uint>* _header;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="links">
39         /// <para>A links.</para>
40         /// <para></para>
41         /// </param>
42         /// <param name="header">
43         /// <para>A header.</para>
44         /// <para></para>
45         /// </param>
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public UInt32UnusedLinksListMethods(RawLink<uint>* links, LinksHeader<uint>* header)
48             : base((byte*)links, (byte*)header)
49         {
50             _links = links;
51             _header = header;
52         }
53
54         /// <summary>
55         /// <para>
56         /// Gets the link reference using the specified link.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         /// <param name="link">
61         /// <para>The link.</para>
62         /// <para></para>
63         /// </param>
64         /// <returns>
65         /// <para>A ref raw link of uint</para>
66         /// <para></para>
67         /// </returns>
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         protected override ref RawLink<uint> GetLinkReference(uint link) => ref _links[link];
70
71         /// <summary>
72         /// <para>
73         /// Gets the header reference.
74         /// </para>
75         /// <para></para>
76         /// </summary>
77         /// <returns>
78         /// <para>A ref links header of uint</para>
79         /// <para></para>
80         /// </returns>
81         [MethodImpl(MethodImplOptions.AggressiveInlining)]
82         protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
83     }
84 }

```

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>
77        /// <para></para>
78        /// </summary>
79        /// <param name="value">
80        /// <para>The value.</para>
81        /// <para></para>
82        /// </param>
83        /// <returns>

```

```

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

```

```

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

```

```

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

```

```

313 [MethodImpl(MethodImplOptions.AggressiveInlining)]
314 protected override ulong Add(ulong first, ulong second) => first + second;
315
316 /// <summary>
317 /// <para>
318 /// Subtracts the first.
319 /// </para>
320 /// <para></para>
321 /// </summary>
322 /// <param name="first">
323 /// <para>The first.</para>
324 /// <para></para>
325 /// </param>
326 /// <param name="second">
327 /// <para>The second.</para>
328 /// <para></para>
329 /// </param>
330 /// <returns>
331 /// <para>The ulong</para>
332 /// <para></para>
333 /// </returns>
334 [MethodImpl(MethodImplOptions.AggressiveInlining)]
335 protected override ulong Subtract(ulong first, ulong second) => first - second;
336
337 /// <summary>
338 /// <para>
339 /// Determines whether this instance first is to the left of second.
340 /// </para>
341 /// <para></para>
342 /// </summary>
343 /// <param name="first">
344 /// <para>The first.</para>
345 /// <para></para>
346 /// </param>
347 /// <param name="second">
348 /// <para>The second.</para>
349 /// <para></para>
350 /// </param>
351 /// <returns>
352 /// <para>The bool</para>
353 /// <para></para>
354 /// </returns>
355 [MethodImpl(MethodImplOptions.AggressiveInlining)]
356 protected override bool FirstIsToLeftOfSecond(ulong first, ulong second)
357 {
358     ref var firstLink = ref Links[first];
359     ref var secondLink = ref Links[second];
360     return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
361         ↪ secondLink.Source, secondLink.Target);
362 }
363
364 /// <summary>
365 /// <para>
366 /// Determines whether this instance first is to the right of second.
367 /// </para>
368 /// <para></para>
369 /// </summary>
370 /// <param name="first">
371 /// <para>The first.</para>
372 /// <para></para>
373 /// </param>
374 /// <param name="second">
375 /// <para>The second.</para>
376 /// <para></para>
377 /// </param>
378 /// <returns>
379 /// <para>The bool</para>
380 /// <para></para>
381 /// </returns>
382 [MethodImpl(MethodImplOptions.AggressiveInlining)]
383 protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
384 {
385     ref var firstLink = ref Links[first];
386     ref var secondLink = ref Links[second];
387     return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
388         ↪ secondLink.Source, secondLink.Target);
389 }

```



```

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>
536 /// <returns>
537 /// <para>A ref links header of ulong</para>
538 /// <para></para>
539 /// </returns>
540 [MethodImpl(MethodImplOptions.AggressiveInlining)]
541 protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;

```

```

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/UInt64LinksRecursionlessSizeBalancedTreeMethod

```

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        /// <summary>
33        /// <para>
34        /// Initializes a new <see cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
35        ↪ instance.
36        /// </para>
37        /// <para></para>
38        /// </summary>
39        /// <param name="constants">
40        /// <para>A constants.</para>
41        /// <para></para>
42        /// </param>
43        /// <param name="links">
44        /// <para>A links.</para>
45        /// <para></para>
46        /// </param>
47        /// <param name="header">
48        /// <para>A header.</para>
49        /// <para></para>
50        /// </param>
51        protected UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
52        ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
53        : base(constants, (byte*)links, (byte*)header)
54        {
55            Links = links;
56            Header = header;
57        }
58    }
59 }

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

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

```

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

```



```

21     /// <para></para>
22     /// </summary>
23     protected new readonly RawLink<ulong>* Links;
24     /// <summary>
25     /// <para>
26     /// The header.
27     /// </para>
28     /// <para></para>
29     /// </summary>
30     protected new readonly LinksHeader<ulong>* Header;
31
32     /// <summary>
33     /// <para>
34     /// Initializes a new <see cref="UInt64LinksSizeBalancedTreeMethodsBase"/> instance.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="constants">
39     /// <para>A constants.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="links">
43     /// <para>A links.</para>
44     /// <para></para>
45     /// </param>
46     /// <param name="header">
47     /// <para>A header.</para>
48     /// <para></para>
49     /// </param>
50     protected UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
51     ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
52     : base(constants, (byte*)links, (byte*)header)
53     {
54         Links = links;
55         Header = header;
56     }
57
58     /// <summary>
59     /// <para>
60     /// Gets the zero.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <returns>
65     /// <para>The ulong</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ulong GetZero() => 0UL;
70
71     /// <summary>
72     /// <para>
73     /// Determines whether this instance equal to zero.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <param name="value">
78     /// <para>The value.</para>
79     /// <para></para>
80     /// </param>
81     /// <returns>
82     /// <para>The bool</para>
83     /// <para></para>
84     /// </returns>
85     [MethodImpl(MethodImplOptions.AggressiveInlining)]
86     protected override bool EqualToZero(ulong value) => value == 0UL;
87
88     /// <summary>
89     /// <para>
90     /// Determines whether this instance are equal.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="first">
95     /// <para>The first.</para>
96     /// <para></para>
97     /// </param>
98     /// <param name="second">

```

```

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

```

```

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

```

```

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

```

```

329    /// <returns>
330    /// <para>The ulong</para>
331    /// <para></para>
332    /// </returns>
333    [MethodImpl(MethodImplOptions.AggressiveInlining)]
334    protected override ulong Subtract(ulong first, ulong second) => first - second;
335
336    /// <summary>
337    /// <para>
338    /// Determines whether this instance first is to the left of second.
339    /// </para>
340    /// <para></para>
341    /// </summary>
342    /// <param name="first">
343    /// <para>The first.</para>
344    /// <para></para>
345    /// </param>
346    /// <param name="second">
347    /// <para>The second.</para>
348    /// <para></para>
349    /// </param>
350    /// <returns>
351    /// <para>The bool</para>
352    /// <para></para>
353    /// </returns>
354    [MethodImpl(MethodImplOptions.AggressiveInlining)]
355    protected override bool FirstIsToLeftOfSecond(ulong first, ulong second)
356    {
357        ref var firstLink = ref Links[first];
358        ref var secondLink = ref Links[second];
359        return FirstIsToLeftOfSecond(firstLink.Source, firstLink.Target,
360            ↪ secondLink.Source, secondLink.Target);
361    }
362
363    /// <summary>
364    /// <para>
365    /// Determines whether this instance first is to the right of second.
366    /// </para>
367    /// <para></para>
368    /// </summary>
369    /// <param name="first">
370    /// <para>The first.</para>
371    /// <para></para>
372    /// </param>
373    /// <param name="second">
374    /// <para>The second.</para>
375    /// <para></para>
376    /// </param>
377    /// <returns>
378    /// <para>The bool</para>
379    /// <para></para>
380    /// </returns>
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>
399    /// </returns>
400    [MethodImpl(MethodImplOptions.AggressiveInlining)]
401    protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;
402
403    /// <summary>
404    /// <para>
405    /// Gets the link reference using the specified link.
406    /// </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)]
54        protected override ref ulong GetLeftReference(ulong node) => ref
55            ↳ Links[node].LeftAsSource;
56
57        /// <summary>
58        /// <para>
59        /// Gets the right reference using the specified node.
60        /// </para>
61        /// <para></para>
62        /// </summary>

```

```

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

```

```

135 [MethodImpl(MethodImplOptions.AggressiveInlining)]
136 protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
    ↳ right;
137
138 /// <summary>
139 /// <para>
140 /// Gets the size using the specified node.
141 /// </para>
142 /// <para></para>
143 /// </summary>
144 /// <param name="node">
145 /// <para>The node.</para>
146 /// <para></para>
147 /// </param>
148 /// <returns>
149 /// <para>The ulong</para>
150 /// <para></para>
151 /// </returns>
152 [MethodImpl(MethodImplOptions.AggressiveInlining)]
153 protected override ulong GetSize(ulong node) => GetSizeValue(Links[node].SizeAsSource);
154
155 /// <summary>
156 /// <para>
157 /// Sets the size using the specified node.
158 /// </para>
159 /// <para></para>
160 /// </summary>
161 /// <param name="node">
162 /// <para>The node.</para>
163 /// <para></para>
164 /// </param>
165 /// <param name="size">
166 /// <para>The size.</para>
167 /// <para></para>
168 /// </param>
169 [MethodImpl(MethodImplOptions.AggressiveInlining)]
170 protected override void SetSize(ulong node, ulong size) => SetSizeValue(ref
    ↳ Links[node].SizeAsSource, size);
171
172 /// <summary>
173 /// <para>
174 /// Determines whether this instance get left is child.
175 /// </para>
176 /// <para></para>
177 /// </summary>
178 /// <param name="node">
179 /// <para>The node.</para>
180 /// <para></para>
181 /// </param>
182 /// <returns>
183 /// <para>The bool</para>
184 /// <para></para>
185 /// </returns>
186 [MethodImpl(MethodImplOptions.AggressiveInlining)]
187 protected override bool GetLeftIsChild(ulong node) =>
    ↳ GetLeftIsChildValue(Links[node].SizeAsSource);
188
189 //[MethodImpl(MethodImplOptions.AggressiveInlining)]
190 //protected override bool GetLeftIsChild(ulong node) => IsChild(node, GetLeft(node));
191
192 /// <summary>
193 /// <para>
194 /// Sets the left is child using the specified node.
195 /// </para>
196 /// <para></para>
197 /// </summary>
198 /// <param name="node">
199 /// <para>The node.</para>
200 /// <para></para>
201 /// </param>
202 /// <param name="value">
203 /// <para>The value.</para>
204 /// <para></para>
205 /// </param>
206 [MethodImpl(MethodImplOptions.AggressiveInlining)]
207 protected override void SetLeftIsChild(ulong node, bool value) =>
    ↳ SetLeftIsChildValue(ref Links[node].SizeAsSource, value);
208

```



```

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

```

```

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

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

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

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

```

```

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     }

```

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>
36         public UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
37             ↪ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
38             ↪ links, header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>

```

```

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>

```

```

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

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

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

/// <summary>
/// <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>
51         /// <para></para>
52         /// </returns>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override ref ulong GetLeftReference(ulong node) => ref
55             ↳ Links[node].LeftAsSource;
56
57         /// <summary>
58         /// <para>

```

```

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

```



```

131     /// <param name="right">
132     /// <para>The right.</para>
133     /// <para></para>
134     /// </param>
135     [MethodImpl(MethodImplOptions.AggressiveInlining)]
136     protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
        ↳ right;

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

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

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

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

201
202     /// <summary>
203     /// <para>
204     /// Determines whether this instance first is to the left of second.
205     /// </para>
206     /// <para></para>

```

```

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

```

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>
67         /// <returns>
68         /// <para>The ref ulong</para>
69         /// <para></para>
70         /// </returns>
71         [MethodImpl(MethodImplOptions.AggressiveInlining)]
72         protected override ref ulong GetRightReference(ulong node) => ref
73             ↳ Links[node].RightAsTarget;
74
75         /// <summary>

```

```

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

```

```

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

```

```

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 :
15         ↳ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
20         ↳ cref="UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="constants">
25         /// <para>A constants.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="links">
29         /// <para>A links.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="header">
33         /// <para>A header.</para>
34         /// <para></para>
35         /// </param>
36         public UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
37         ↳ constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
38         ↳ links, header) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the left reference using the specified node.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="node">
47         /// <para>The node.</para>
48         /// <para></para>
49         /// </param>
50         /// <returns>
51         /// <para>The ref ulong</para>
52         /// <para></para>
53         /// </returns>
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         protected override ref ulong GetLeftReference(ulong node) => ref
56         ↳ Links[node].LeftAsTarget;
57
58         /// <summary>
59         /// <para>
60         /// Gets the right reference using the specified node.
61         /// </para>
62         /// <para></para>
63         /// </summary>
64         /// <param name="node">
65         /// <para>The node.</para>
66         /// <para></para>
67         /// </param>
68         /// <returns>
69         /// <para>The ref ulong</para>
70         /// <para></para>
71         /// </returns>
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         protected override ref ulong GetRightReference(ulong node) => ref
74         ↳ Links[node].RightAsTarget;
75     }
76 }

```



```

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 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 }

```

1.106 ./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>
70        [MethodImpl(MethodImplOptions.AggressiveInlining)]
71        protected override ref ulong GetRightReference(ulong node) => ref
72            ↪ Links[node].RightAsTarget;
73
74        /// <summary>
75        /// <para>
76        /// Gets the left using the specified node.
77        /// </para>
78        /// </summary>
79        /// <param name="node">
80        /// <para>The node.</para>
81        /// <para></para>
82        /// </param>
83        /// <returns>
84        /// <para>The ref ulong</para>
85        /// <para></para>
86        /// </returns>
87        [MethodImpl(MethodImplOptions.AggressiveInlining)]
88        protected override ref ulong GetLeftUsingNode(ulong node) => ref
89            ↪ Links[node].LeftUsingNode;
90
91        /// <summary>
92        /// <para>
93        /// Gets the right using the specified node.
94        /// </para>
95        /// </summary>
96        /// <param name="node">
97        /// <para>The node.</para>
98        /// <para></para>
99        /// </param>
100       /// <returns>
101       /// <para>The ref ulong</para>
102       /// <para></para>
103       /// </returns>
104       [MethodImpl(MethodImplOptions.AggressiveInlining)]
105       protected override ref ulong GetRightUsingNode(ulong node) => ref
106           ↪ Links[node].RightUsingNode;
107
108       /// <summary>
109       /// <para>
110       /// Gets the left using the specified node.
111       /// </para>
112       /// </summary>
113       /// <param name="node">
114       /// <para>The node.</para>
115       /// <para></para>
116       /// </param>
117       /// <returns>
118       /// <para>The ref ulong</para>
119       /// <para></para>
120       /// </returns>
121       [MethodImpl(MethodImplOptions.AggressiveInlining)]
122       protected override ref ulong GetLeftUsingNodeAndHeader(ulong node) => ref
123           ↪ Links[node].LeftUsingNodeAndHeader;
124
125       /// <summary>
126       /// <para>
127       /// Gets the right using the specified node.
128       /// </para>
129       /// </summary>
130       /// <param name="node">
131       /// <para>The node.</para>
132       /// <para></para>
133       /// </param>
134       /// <returns>
135       /// <para>The ref ulong</para>
136       /// <para></para>
137       /// </returns>
138       [MethodImpl(MethodImplOptions.AggressiveInlining)]
139       protected override ref ulong GetRightUsingNodeAndHeader(ulong node) => ref
140           ↪ Links[node].RightUsingNodeAndHeader;
141
142       /// <summary>
143       /// <para>
144       /// Gets the left using the specified node.
145       /// </para>
146       /// </summary>
147       /// <param name="node">
148       /// <para>The node.</para>
149       /// <para></para>
150       /// </param>
151       /// <returns>
152       /// <para>The ref ulong</para>
153       /// <para></para>
154       /// </returns>
155       [MethodImpl(MethodImplOptions.AggressiveInlining)]
156       protected override ref ulong GetLeftUsingNodeAndHeaderAndHeader(ulong node) => ref
157           ↪ Links[node].LeftUsingNodeAndHeaderAndHeader;
158
159       /// <summary>
160       /// <para>
161       /// Gets the right using the specified node.
162       /// </para>
163       /// </summary>
164       /// <param name="node">
165       /// <para>The node.</para>
166       /// <para></para>
167       /// </param>
168       /// <returns>
169       /// <para>The ref ulong</para>
170       /// <para></para>
171       /// </returns>
172       [MethodImpl(MethodImplOptions.AggressiveInlining)]
173       protected override ref ulong GetRightUsingNodeAndHeaderAndHeader(ulong node) => ref
174           ↪ Links[node].RightUsingNodeAndHeaderAndHeader;
175
176       /// <summary>
177       /// <para>
178       /// Gets the left using the specified node.
179       /// </para>
180       /// </summary>
181       /// <param name="node">
182       /// <para>The node.</para>
183       /// <para></para>
184       /// </param>
185       /// <returns>
186       /// <para>The ref ulong</para>
187       /// <para></para>
188       /// </returns>
189       [MethodImpl(MethodImplOptions.AggressiveInlining)]
190       protected override ref ulong GetLeftUsingNodeAndHeaderAndHeaderAndHeader(ulong node) => ref
191           ↪ Links[node].LeftUsingNodeAndHeaderAndHeaderAndHeader;
192
193       /// <summary>
194       /// <para>
195       /// Gets the right using the specified node.
196       /// </para>
197       /// </summary>
198       /// <param name="node">
199       /// <para>The node.</para>
200       /// <para></para>
201       /// </param>
202       /// <returns>
203       /// <para>The ref ulong</para>
204       /// <para></para>
205       /// </returns>
206       [MethodImpl(MethodImplOptions.AggressiveInlining)]
207       protected override ref ulong GetRightUsingNodeAndHeaderAndHeaderAndHeader(ulong node) => ref
208           ↪ Links[node].RightUsingNodeAndHeaderAndHeaderAndHeader;
209
210       /// <summary>
211       /// <para>
212       /// Gets the left using the specified node.
213       /// </para>
214       /// </summary>
215       /// <param name="node">
216       /// <para>The node.</para>
217       /// <para></para>
218       /// </param>
219       /// <returns>
220       /// <para>The ref ulong</para>
221       /// <para></para>
222       /// </returns>
223       [MethodImpl(MethodImplOptions.AggressiveInlining)]
224       protected override ref ulong GetLeftUsingNodeAndHeaderAndHeaderAndHeaderAndHeader(ulong node) => ref
225           ↪ Links[node].LeftUsingNodeAndHeaderAndHeaderAndHeaderAndHeader;
226
227       /// <summary>
228       /// <para>
229       /// Gets the right using the specified node.
230       /// </para>
231       /// </summary>
232       /// <param name="node">
233       /// <para>The node.</para>
234       /// <para></para>
235       /// </param>
236       /// <returns>
237       /// <para>The ref ulong</para>
238       /// <para></para>
239       /// </returns>
240       [MethodImpl(MethodImplOptions.AggressiveInlining)]
241       protected override ref ulong GetRightUsingNodeAndHeaderAndHeaderAndHeaderAndHeader(ulong node) => ref
242           ↪ Links[node].RightUsingNodeAndHeaderAndHeaderAndHeaderAndHeader;
243
244       /// <summary>
245       /// <para>
246       /// Gets the left using the specified node.
247       /// </para>
248       /// </summary>
249       /// <param name="node">
250       /// <para>The node.</para>
251       /// <para></para>
252       /// </param>
253       /// <returns>
254       /// <para>The ref ulong</para>
255       /// <para></para>
256       /// </returns>
257       [MethodImpl(MethodImplOptions.AggressiveInlining)]
258       protected override ref ulong GetLeftUsingNodeAndHeaderAndHeaderAndHeaderAndHeaderAndHeader(ulong node) => ref
259           ↪ Links[node].LeftUsingNodeAndHeaderAndHeaderAndHeaderAndHeaderAndHeader;
260
261       /// <summary>
262       /// <para>
263       /// Gets the right using the specified node.
264       /// </para>
265       /// </summary>
266       /// <param name="node">
267       /// <para>The node.</para>
268       /// <para></para>
269       /// </param>
270       /// <returns>
271       /// <para>The ref ulong</para>
272       /// <para></para>
273       /// </returns>
274       [MethodImpl(MethodImplOptions.AggressiveInlining)]
275       protected override ref ulong GetRightUsingNodeAndHeaderAndHeaderAndHeaderAndHeaderAndHeader(ulong node) => ref
276           ↪ Links[node].RightUsingNodeAndHeaderAndHeaderAndHeaderAndHeaderAndHeader;
277
278       /// <summary>
279       /// <para>
280       /// Gets the left using the specified node.
281       /// </para>
282       /// </summary>
283       /// <param name="node">
284       /// <para>The node.</para>
285       /// <para></para>
286       /// </param>
287       /// <returns>
288       /// <para>The ref ulong</para>
```

```

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

```

```

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

```

```

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

```

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>

```

```

14  /// </summary>
15  public unsafe class UInt64UnitedMemoryLinks : UnitedMemoryLinksBase<ulong>
16  {
17      /// <summary>
18      /// <para>
19      /// The create source tree methods.
20      /// </para>
21      /// <para></para>
22      /// </summary>
23      private readonly Func<ILinksTreeMethods<ulong>> _createSourceTreeMethods;
24      /// <summary>
25      /// <para>
26      /// The create target tree methods.
27      /// </para>
28      /// <para></para>
29      /// </summary>
30      private readonly Func<ILinksTreeMethods<ulong>> _createTargetTreeMethods;
31      /// <summary>
32      /// <para>
33      /// The header.
34      /// </para>
35      /// <para></para>
36      /// </summary>
37      private LinksHeader<ulong>* _header;
38      /// <summary>
39      /// <para>
40      /// The links.
41      /// </para>
42      /// <para></para>
43      /// </summary>
44      private RawLink<ulong>* _links;
45
46      /// <summary>
47      /// <para>
48      /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
49      /// </para>
50      /// <para></para>
51      /// </summary>
52      /// <param name="address">
53      /// <para>A address.</para>
54      /// <para></para>
55      /// </param>
56      [MethodImpl(MethodImplOptions.AggressiveInlining)]
57      public UInt64UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
58
59      /// <summary>
60      /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
61      ↪ минимальным шагом расширения базы данных.
62      /// </summary>
63      /// <param name="address">Полный путь к файлу базы данных.</param>
64      /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
65      ↪ байтах.</param>
66      [MethodImpl(MethodImplOptions.AggressiveInlining)]
67      public UInt64UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
68      ↪ FileMappedResizableDirectMemory(address, memoryReservationStep),
69      ↪ memoryReservationStep) { }
70
71      /// <summary>
72      /// <para>
73      /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
74      /// </para>
75      /// <para></para>
76      /// </summary>
77      /// <param name="memory">
78      /// <para>A memory.</para>
79      /// <para></para>
80      /// </param>
81      [MethodImpl(MethodImplOptions.AggressiveInlining)]
82      public UInt64UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
83      ↪ DefaultLinksSizeStep) { }
84
85      /// <summary>
86      /// <para>
87      /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
88      /// </para>
89      /// <para></para>
90      /// </summary>
91      /// <param name="memory">

```



```

87     /// <para>A memory.</para>
88     /// <para></para>
89     /// </param>
90     /// <param name="memoryReservationStep">
91     /// <para>A memory reservation step.</para>
92     /// <para></para>
93     /// </param>
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep) : this(memory, memoryReservationStep,
        ↳ Default<LinksConstants<ulong>>.Instance, IndexTreeType.Default) { }

96
97     /// <summary>
98     /// <para>
99     /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
100    /// </para>
101    /// <para></para>
102    /// </summary>
103    /// <param name="memory">
104    /// <para>A memory.</para>
105    /// <para></para>
106    /// </param>
107    /// <param name="memoryReservationStep">
108    /// <para>A memory reservation step.</para>
109    /// <para></para>
110    /// </param>
111    /// <param name="constants">
112    /// <para>A constants.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="indexTreeType">
116    /// <para>A index tree type.</para>
117    /// <para></para>
118    /// </param>
119    [MethodImpl(MethodImplOptions.AggressiveInlining)]
120    public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
        ↳ memoryReservationStep, LinksConstants<ulong> constants, IndexTreeType indexTreeType)
        ↳ : base(memory, memoryReservationStep, constants)
121    {
122        if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
123        {
124            _createSourceTreeMethods = () => new
125                ↳ UInt64LinksSourcesAvlBalancedTreeMethods(Constants, _links, _header);
126            _createTargetTreeMethods = () => new
127                ↳ UInt64LinksTargetsAvlBalancedTreeMethods(Constants, _links, _header);
128        }
129        else if (indexTreeType == IndexTreeType.SizeBalancedTree)
130        {
131            _createSourceTreeMethods = () => new
132                ↳ UInt64LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
133            _createTargetTreeMethods = () => new
134                ↳ UInt64LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
135        }
136        else
137        {
138            _createSourceTreeMethods = () => new
139                ↳ UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
140                ↳ _header);
141            _createTargetTreeMethods = () => new
142                ↳ UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
143                ↳ _header);
144        }
145        Init(memory, memoryReservationStep);
146    }

147    /// <summary>
148    /// <para>
149    /// Sets the pointers using the specified memory.
150    /// </para>
151    /// <para></para>
152    /// </summary>
153    /// <param name="memory">
154    /// <para>The memory.</para>
155    /// <para></para>
156    /// </param>
157    [MethodImpl(MethodImplOptions.AggressiveInlining)]
158    protected override void SetPointers(IResizableDirectMemory memory)

```

```

152 {
153     _header = (LinksHeader<ulong>*)memory.Pointer;
154     _links = (RawLink<ulong>*)memory.Pointer;
155     SourcesTreeMethods = _createSourceTreeMethods();
156     TargetsTreeMethods = _createTargetTreeMethods();
157     UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
158 }
159
160 /// <summary>
161 /// <para>
162 /// Resets the pointers.
163 /// </para>
164 /// <para></para>
165 /// </summary>
166 [MethodImpl(MethodImplOptions.AggressiveInlining)]
167 protected override void ResetPointers()
168 {
169     base.ResetPointers();
170     _links = null;
171     _header = null;
172 }
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 ulong</para>
182 /// <para></para>
183 /// </returns>
184 [MethodImpl(MethodImplOptions.AggressiveInlining)]
185 protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
186
187 /// <summary>
188 /// <para>
189 /// Gets the link reference using the specified link index.
190 /// </para>
191 /// <para></para>
192 /// </summary>
193 /// <param name="linkIndex">
194 /// <para>The link index.</para>
195 /// <para></para>
196 /// </param>
197 /// <returns>
198 /// <para>A ref raw link of ulong</para>
199 /// <para></para>
200 /// </returns>
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
    ↪ _links[linkIndex];
203
204 /// <summary>
205 /// <para>
206 /// Determines whether this instance are equal.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 /// <param name="first">
211 /// <para>The first.</para>
212 /// <para></para>
213 /// </param>
214 /// <param name="second">
215 /// <para>The second.</para>
216 /// <para></para>
217 /// </param>
218 /// <returns>
219 /// <para>The bool</para>
220 /// <para></para>
221 /// </returns>
222 [MethodImpl(MethodImplOptions.AggressiveInlining)]
223 protected override bool AreEqual(ulong first, ulong second) => first == second;
224
225 /// <summary>
226 /// <para>
227 /// Determines whether this instance less than.
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 LessThan(ulong first, ulong second) => first < second;
245
246    /// <summary>
247    /// <para>
248    /// Determines whether this instance less or equal than.
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 LessOrEqualThan(ulong first, ulong second) => first <= second;
266
267    /// <summary>
268    /// <para>
269    /// Determines whether this instance greater than.
270    /// </para>
271    /// <para></para>
272    /// </summary>
273    /// <param name="first">
274    /// <para>The first.</para>
275    /// <para></para>
276    /// </param>
277    /// <param name="second">
278    /// <para>The second.</para>
279    /// <para></para>
280    /// </param>
281    /// <returns>
282    /// <para>The bool</para>
283    /// <para></para>
284    /// </returns>
285    [MethodImpl(MethodImplOptions.AggressiveInlining)]
286    protected override bool GreaterThan(ulong first, ulong second) => first > second;
287
288    /// <summary>
289    /// <para>
290    /// Determines whether this instance greater or equal than.
291    /// </para>
292    /// <para></para>
293    /// </summary>
294    /// <param name="first">
295    /// <para>The first.</para>
296    /// <para></para>
297    /// </param>
298    /// <param name="second">
299    /// <para>The second.</para>
300    /// <para></para>
301    /// </param>
302    /// <returns>
303    /// <para>The bool</para>
304    /// <para></para>
305    /// </returns>
306    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

307     protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
308
309     /// <summary>
310     /// <para>
311     /// Gets the zero.
312     /// </para>
313     /// <para></para>
314     /// </summary>
315     /// <returns>
316     /// <para>The ulong</para>
317     /// <para></para>
318     /// </returns>
319     [MethodImpl(MethodImplOptions.AggressiveInlining)]
320     protected override ulong GetZero() => 0UL;
321
322     /// <summary>
323     /// <para>
324     /// Gets the one.
325     /// </para>
326     /// <para></para>
327     /// </summary>
328     /// <returns>
329     /// <para>The ulong</para>
330     /// <para></para>
331     /// </returns>
332     [MethodImpl(MethodImplOptions.AggressiveInlining)]
333     protected override ulong GetOne() => 1UL;
334
335     /// <summary>
336     /// <para>
337     /// Converts the to int 64 using the specified value.
338     /// </para>
339     /// <para></para>
340     /// </summary>
341     /// <param name="value">
342     /// <para>The value.</para>
343     /// <para></para>
344     /// </param>
345     /// <returns>
346     /// <para>The long</para>
347     /// <para></para>
348     /// </returns>
349     [MethodImpl(MethodImplOptions.AggressiveInlining)]
350     protected override long ConvertToInt64(ulong value) => (long)value;
351
352     /// <summary>
353     /// <para>
354     /// Converts the to address using the specified value.
355     /// </para>
356     /// <para></para>
357     /// </summary>
358     /// <param name="value">
359     /// <para>The value.</para>
360     /// <para></para>
361     /// </param>
362     /// <returns>
363     /// <para>The ulong</para>
364     /// <para></para>
365     /// </returns>
366     [MethodImpl(MethodImplOptions.AggressiveInlining)]
367     protected override ulong ConvertToAddress(long value) => (ulong)value;
368
369     /// <summary>
370     /// <para>
371     /// Adds the first.
372     /// </para>
373     /// <para></para>
374     /// </summary>
375     /// <param name="first">
376     /// <para>The first.</para>
377     /// <para></para>
378     /// </param>
379     /// <param name="second">
380     /// <para>The second.</para>
381     /// <para></para>
382     /// </param>
383     /// <returns>
384     /// <para>The ulong</para>

```

```

385     /// <para></para>
386     /// </returns>
387     [MethodImpl(MethodImplOptions.AggressiveInlining)]
388     protected override ulong Add(ulong first, ulong second) => first + second;
389
390     /// <summary>
391     /// <para>
392     /// Subtracts the first.
393     /// </para>
394     /// <para></para>
395     /// </summary>
396     /// <param name="first">
397     /// <para>The first.</para>
398     /// <para></para>
399     /// </param>
400     /// <param name="second">
401     /// <para>The second.</para>
402     /// <para></para>
403     /// </param>
404     /// <returns>
405     /// <para>The ulong</para>
406     /// <para></para>
407     /// </returns>
408     [MethodImpl(MethodImplOptions.AggressiveInlining)]
409     protected override ulong Subtract(ulong first, ulong second) => first - second;
410
411     /// <summary>
412     /// <para>
413     /// Increments the link.
414     /// </para>
415     /// <para></para>
416     /// </summary>
417     /// <param name="link">
418     /// <para>The link.</para>
419     /// <para></para>
420     /// </param>
421     /// <returns>
422     /// <para>The ulong</para>
423     /// <para></para>
424     /// </returns>
425     [MethodImpl(MethodImplOptions.AggressiveInlining)]
426     protected override ulong Increment(ulong link) => ++link;
427
428     /// <summary>
429     /// <para>
430     /// Decrements the link.
431     /// </para>
432     /// <para></para>
433     /// </summary>
434     /// <param name="link">
435     /// <para>The link.</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 Decrement(ulong link) => --link;
444 }
445 }

```

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     /// <summary>
18     /// <para>
19     /// The links.
20     /// </para>
21     /// <para></para>
22     /// </summary>
23     private readonly RawLink<ulong>* _links;
24     /// <summary>
25     /// <para>
26     /// The header.
27     /// </para>
28     /// <para></para>
29     /// </summary>
30     private readonly LinksHeader<ulong>* _header;
31
32     /// <summary>
33     /// <para>
34     /// Initializes a new <see cref="UInt64UnusedLinksListMethods"/> instance.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <param name="links">
39     /// <para>A links.</para>
40     /// <para></para>
41     /// </param>
42     /// <param name="header">
43     /// <para>A header.</para>
44     /// <para></para>
45     /// </param>
46     [MethodImpl(MethodImplOptions.AggressiveInlining)]
47     public UInt64UnusedLinksListMethods(RawLink<ulong>* links, LinksHeader<ulong>* header)
48         : base((byte*)links, (byte*)header)
49     {
50         _links = links;
51         _header = header;
52     }
53
54     /// <summary>
55     /// <para>
56     /// Gets the link reference using the specified link.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     /// <param name="link">
61     /// <para>The link.</para>
62     /// <para></para>
63     /// </param>
64     /// <returns>
65     /// <para>A ref raw link of ulong</para>
66     /// <para></para>
67     /// </returns>
68     [MethodImpl(MethodImplOptions.AggressiveInlining)]
69     protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref _links[link];
70
71     /// <summary>
72     /// <para>
73     /// Gets the header reference.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     /// <returns>
78     /// <para>A ref links header of ulong</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
83 }
84 }

```

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

```

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

```

```

9      /// <summary>
10     /// <para>
11     /// Represents the properties operator.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="LinksOperatorBase{TLink}"/>
16     /// <seealso cref="IProperties{TLink, TLink, TLink}"/>
17     public class PropertiesOperator<TLink> : LinksOperatorBase<TLink>, IProperties<TLink, TLink,
    ↪ TLink>
18     {
19         /// <summary>
20         /// <para>
21         /// The default.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         private static readonly EqualityComparer<TLink> _equalityComparer =
    ↪ EqualityComparer<TLink>.Default;
26
27         /// <summary>
28         /// <para>
29         /// Initializes a new <see cref="PropertiesOperator"/> instance.
30         /// </para>
31         /// <para></para>
32         /// </summary>
33         /// <param name="links">
34         /// <para>A links.</para>
35         /// <para></para>
36         /// </param>
37         [MethodImpl(MethodImplOptions.AggressiveInlining)]
38         public PropertiesOperator(ILinks<TLink> links) : base(links) { }
39
40         /// <summary>
41         /// <para>
42         /// Gets the value using the specified object.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="@object">
47         /// <para>The object.</para>
48         /// <para></para>
49         /// </param>
50         /// <param name="property">
51         /// <para>The property.</para>
52         /// <para></para>
53         /// </param>
54         /// <returns>
55         /// <para>The link</para>
56         /// <para></para>
57         /// </returns>
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         public TLink GetValue(TLink @object, TLink property)
60         {
61             var links = _links;
62             var objectProperty = links.SearchOrDefault(@object, property);
63             if (_equalityComparer.Equals(objectProperty, default))
64             {
65                 return default;
66             }
67             var constants = links.Constants;
68             var any = constants.Any;
69             var query = new Link<TLink>(any, objectProperty, any);
70             var valueLink = links.SingleOrDefault(query);
71             if (valueLink == null)
72             {
73                 return default;
74             }
75             return links.GetTarget(valueLink[constants.IndexPart]);
76         }
77
78         /// <summary>
79         /// <para>
80         /// Sets the value using the specified object.
81         /// </para>
82         /// <para></para>
83         /// </summary>
84         /// <param name="@object">

```

```

85     /// <para>The object.</para>
86     /// <para></para>
87     /// </param>
88     /// <param name="property">
89     /// <para>The property.</para>
90     /// <para></para>
91     /// </param>
92     /// <param name="value">
93     /// <para>The value.</para>
94     /// <para></para>
95     /// </param>
96     [MethodImpl(MethodImplOptions.AggressiveInlining)]
97     public void SetValue(TLink @object, TLink property, TLink value)
98     {
99         var links = _links;
100         var objectProperty = links.GetOrCreate(@object, property);
101         links.DeleteMany(links.AllIndices(links.Constants.Any, objectProperty));
102         links.GetOrCreate(objectProperty, value);
103     }
104 }
105 }

```

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         /// <summary>
20         /// <para>
21         /// The default.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         private static readonly EqualityComparer<TLink> _equalityComparer =
26             ↪ EqualityComparer<TLink>.Default;
27
28         /// <summary>
29         /// <para>
30         /// The property marker.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         private readonly TLink _propertyMarker;
35
36         /// <summary>
37         /// <para>
38         /// The property value marker.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         private readonly TLink _propertyValueMarker;
43
44         /// <summary>
45         /// <para>
46         /// Initializes a new <see cref="PropertyOperator" /> instance.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         /// <param name="links">
51         /// <para>A links.</para>
52         /// <para></para>
53         /// </param>
54         /// <param name="propertyMarker">
55         /// <para>A property marker.</para>
56         /// <para></para>
57         /// </param>

```



```

56     /// <param name="propertyValueMarker">
57     /// <para>A property value marker.</para>
58     /// </para>
59     /// </param>
60     [MethodImpl(MethodImplOptions.AggressiveInlining)]
61     public PropertyOperator(ILinks<TLink> links, TLink propertyMarker, TLink
    ↪     propertyValueMarker) : base(links)
62     {
63         _propertyMarker = propertyMarker;
64         _propertyValueMarker = propertyValueMarker;
65     }
66
67     /// <summary>
68     /// <para>
69     /// Gets the link.
70     /// </para>
71     /// <para></para>
72     /// </summary>
73     /// <param name="link">
74     /// <para>The link.</para>
75     /// <para></para>
76     /// </param>
77     /// <returns>
78     /// <para>The link</para>
79     /// <para></para>
80     /// </returns>
81     [MethodImpl(MethodImplOptions.AggressiveInlining)]
82     public TLink Get(TLink link)
83     {
84         var property = _links.SearchOrDefault(link, _propertyMarker);
85         return GetValue(GetContainer(property));
86     }
87
88     /// <summary>
89     /// <para>
90     /// Gets the container using the specified property.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <param name="property">
95     /// <para>The property.</para>
96     /// <para></para>
97     /// </param>
98     /// <returns>
99     /// <para>The value container.</para>
100    /// <para></para>
101    /// </returns>
102    [MethodImpl(MethodImplOptions.AggressiveInlining)]
103    private TLink GetContainer(TLink property)
104    {
105        var valueContainer = default(TLink);
106        if (_equalityComparer.Equals(property, default))
107        {
108            return valueContainer;
109        }
110        var links = _links;
111        var constants = links.Constants;
112        var countinueConstant = constants.Continue;
113        var breakConstant = constants.Break;
114        var anyConstant = constants.Any;
115        var query = new Link<TLink>(anyConstant, property, anyConstant);
116        links.Each(candidate =>
117        {
118            var candidateTarget = links.GetTarget(candidate);
119            var valueTarget = links.GetTarget(candidateTarget);
120            if (_equalityComparer.Equals(valueTarget, _propertyValueMarker))
121            {
122                valueContainer = links.GetIndex(candidate);
123                return breakConstant;
124            }
125            return countinueConstant;
126        }, query);
127        return valueContainer;
128    }
129
130    /// <summary>
131    /// <para>
132    /// Gets the value using the specified container.
133    /// </para>

```

```

134     /// <para></para>
135     /// </summary>
136     /// <param name="container">
137     /// <para>The container.</para>
138     /// <para></para>
139     /// </param>
140     /// <returns>
141     /// <para>The link</para>
142     /// <para></para>
143     /// </returns>
144     [MethodImpl(MethodImplOptions.AggressiveInlining)]
145     private TLink GetValue(TLink container) => _equalityComparer.Equals(container, default)
        ↪ ? default : _links.GetTarget(container);
146
147     /// <summary>
148     /// <para>
149     /// Sets the link.
150     /// </para>
151     /// <para></para>
152     /// </summary>
153     /// <param name="link">
154     /// <para>The link.</para>
155     /// <para></para>
156     /// </param>
157     /// <param name="value">
158     /// <para>The value.</para>
159     /// <para></para>
160     /// </param>
161     [MethodImpl(MethodImplOptions.AggressiveInlining)]
162     public void Set(TLink link, TLink value)
163     {
164         var links = _links;
165         var property = links.GetOrCreate(link, _propertyMarker);
166         var container = GetContainer(property);
167         if (_equalityComparer.Equals(container, default))
168         {
169             links.GetOrCreate(property, value);
170         }
171         else
172         {
173             links.Update(container, property, value);
174         }
175     }
176 }
177 }

```

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         /// <summary>
20         /// <para>
21         /// The default.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
26
27         /// <summary>
28         /// <para>
29         /// The stack.
30         /// </para>
31         /// <para></para>

```

```

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

```

```

110     /// <para>The element.</para>
111     /// <para></para>
112     /// </returns>
113     [MethodImpl(MethodImplOptions.AggressiveInlining)]
114     public TLink Pop()
115     {
116         var element = Peek();
117         if (!_equalityComparer.Equals(element, _stack))
118         {
119             var top = GetTop();
120             var previousTop = _links.GetSource(top);
121             _links.Update(_stack, GetStackMarker(), previousTop);
122             _links.Delete(top);
123         }
124         return element;
125     }
126
127     /// <summary>
128     /// <para>
129     /// Pushes the element.
130     /// </para>
131     /// <para></para>
132     /// </summary>
133     /// <param name="element">
134     /// <para>The element.</para>
135     /// <para></para>
136     /// </param>
137     [MethodImpl(MethodImplOptions.AggressiveInlining)]
138     public void Push(TLink element) => _links.Update(_stack, GetStackMarker(),
139         ↪ _links.GetOrCreate(GetTop(), element));
140 }

```

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
    ↳ ReaderWriterLockSynchronization(), links) { }
78
79 /// <summary>
80 /// <para>
81 /// Initializes a new <see cref="SynchronizedLinks"/> instance.
82 /// </para>
83 /// <para></para>
84 /// </summary>
85 /// <param name="synchronization">
86 /// <para>A synchronization.</para>
87 /// <para></para>
88 /// </param>
89 /// <param name="links">
90 /// <para>A links.</para>
91 /// <para></para>
92 /// </param>
93 [MethodImpl(MethodImplOptions.AggressiveInlining)]
94 public SynchronizedLinks(ISynchronization synchronization, ILinks<TLinkAddress> links)
95 {
96     SyncRoot = synchronization;
97     Sync = this;
98     Unsync = links;
99     Constants = links.Constants;
100 }
101
102 /// <summary>
103 /// <para>
104 /// Counts the restriction.
105 /// </para>
106 /// <para></para>
107 /// </summary>
108 /// <param name="restriction">
109 /// <para>The restriction.</para>
110 /// <para></para>
111 /// </param>
112 /// <returns>
113 /// <para>The link address</para>
114 /// <para></para>
115 /// </returns>
116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 public TLinkAddress Count(IList<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<IList<TLinkAddress>, TLinkAddress> handler,
    ↳ IList<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(IList<TLinkAddress> restrictions) =>
156         ↳ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Create);
157
158     /// <summary>
159     /// <para>
160     /// Updates the restrictions.
161     /// </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(IList<TLinkAddress> restrictions, IList<TLinkAddress>
177         ↳ substitution) => SyncRoot.ExecuteWriteOperation(restrictions, substitution,
178         ↳ Unsync.Update);
179
180     /// <summary>
181     /// <para>
182     /// Deletes the restrictions.
183     /// </para>
184     /// </summary>
185     /// <param name="restrictions">
186     /// <para>The restrictions.</para>
187     /// <para></para>
188     /// </param>
189     [MethodImpl(MethodImplOptions.AggressiveInlining)]
190     public void Delete(IList<TLinkAddress> restrictions) =>
191         ↳ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Delete);
192
193     //public T Trigger(IList<T> restriction, Func<IList<T>, IList<T>, T> matchedHandler,
194     //↳ IList<T> substitution, Func<IList<T>, IList<T>, T> substitutedHandler)
195     //{
196     //    if (restriction != null && substitution != null &&
197     //↳ !substitution.EqualTo(restriction))
198     //        return SyncRoot.ExecuteWriteOperation(restriction, matchedHandler,
199     //↳ substitution, substitutedHandler, Unsync.Trigger);
200     //    return SyncRoot.ExecuteReadOperation(restriction, matchedHandler, substitution,
201     //↳ substitutedHandler, Unsync.Trigger);
202     //}
203 }
204 }

```

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

```

1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5  using Platform.Singletons;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the int 64 links extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class UInt64LinksExtensions
18     {
19         /// <summary>

```

```

20    /// <para>
21    /// The instance.
22    /// </para>
23    /// <para></para>
24    /// </summary>
25    public static readonly LinksConstants<ulong> Constants =
26    ↪ Default<LinksConstants<ulong>>.Instance;
27
28    /// <summary>
29    /// <para>
30    /// Determines whether any link is any.
31    /// </para>
32    /// <para></para>
33    /// </summary>
34    /// <param name="links">
35    /// <para>The links.</para>
36    /// <para></para>
37    /// </param>
38    /// <param name="sequence">
39    /// <para>The sequence.</para>
40    /// <para></para>
41    /// </param>
42    /// <returns>
43    /// <para>The bool</para>
44    /// <para></para>
45    /// </returns>
46    [MethodImpl(MethodImplOptions.AggressiveInlining)]
47    public static bool AnyLinkIsAny(this ILinks<ulong> links, params ulong[] sequence)
48    {
49        if (sequence == null)
50        {
51            return false;
52        }
53        var constants = links.Constants;
54        for (var i = 0; i < sequence.Length; i++)
55        {
56            if (sequence[i] == constants.Any)
57            {
58                return true;
59            }
60        }
61        return false;
62    }
63
64    /// <summary>
65    /// <para>
66    /// Formats the structure using the specified links.
67    /// </para>
68    /// <para></para>
69    /// </summary>
70    /// <param name="links">
71    /// <para>The links.</para>
72    /// <para></para>
73    /// </param>
74    /// <param name="linkIndex">
75    /// <para>The link index.</para>
76    /// <para></para>
77    /// </param>
78    /// <param name="isElement">
79    /// <para>The is element.</para>
80    /// <para></para>
81    /// </param>
82    /// <param name="renderIndex">
83    /// <para>The render index.</para>
84    /// <para></para>
85    /// </param>
86    /// <param name="renderDebug">
87    /// <para>The render debug.</para>
88    /// <para></para>
89    /// </param>
90    /// <returns>
91    /// <para>The string</para>
92    /// <para></para>
93    /// </returns>
94    [MethodImpl(MethodImplOptions.AggressiveInlining)]
95    public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
96    ↪ Func<Link<ulong>, bool> isElement, bool renderIndex = false, bool renderDebug =
97    ↪ false)

```



```

95     {
96         var sb = new StringBuilder();
97         var visited = new HashSet<ulong>();
98         links.AppendStructure(sb, visited, linkIndex, isElement, (innerSb, link) =>
99             ↪ innerSb.Append(link.Index), renderIndex, renderDebug);
100         return sb.ToString();
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,
138         ↪ Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
139         ↪ bool renderIndex = false, bool renderDebug = false)
140     {
141         var sb = new StringBuilder();
142         var visited = new HashSet<ulong>();
143         links.AppendStructure(sb, visited, linkIndex, isElement, appendElement, renderIndex,
144             ↪ renderDebug);
145         return sb.ToString();
146     }
147     /// <summary>
148     /// <para>
149     /// Appends the structure using the specified links.
150     /// </para>
151     /// <para></para>
152     /// </summary>
153     /// <param name="links">
154     /// <para>The links.</para>
155     /// <para></para>
156     /// </param>
157     /// <param name="sb">
158     /// <para>The sb.</para>
159     /// <para></para>
160     /// </param>
161     /// <param name="visited">
162     /// <para>The visited.</para>
163     /// <para></para>
164     /// </param>
165     /// <param name="linkIndex">
166     /// <para>The link index.</para>
167     /// <para></para>
168     /// </param>
169     /// <param name="isElement">

```

```

168 /// <para>The is element.</para>
169 /// <para></para>
170 /// </param>
171 /// <param name="appendElement">
172 /// <para>The append element.</para>
173 /// <para></para>
174 /// </param>
175 /// <param name="renderIndex">
176 /// <para>The render index.</para>
177 /// <para></para>
178 /// </param>
179 /// <param name="renderDebug">
180 /// <para>The render debug.</para>
181 /// <para></para>
182 /// </param>
183 /// <exception cref="ArgumentNullException">
184 /// <para></para>
185 /// <para></para>
186 /// </exception>
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 public static void AppendStructure(this ILinks<ulong> links, StringBuilder sb,
    ↳ HashSet<ulong> visited, ulong linkIndex, Func<Link<ulong>, bool> isElement,
    ↳ Action<StringBuilder, Link<ulong>> appendElement, bool renderIndex = false, bool
    ↳ renderDebug = false)
189 {
190     if (sb == null)
191     {
192         throw new ArgumentNullException(nameof(sb));
193     }
194     if (linkIndex == Constants.Null || linkIndex == Constants.Any || linkIndex ==
    ↳ Constants.Itself)
195     {
196         return;
197     }
198     if (links.Exists(linkIndex))
199     {
200         if (visited.Add(linkIndex))
201         {
202             sb.Append('(');
203             var link = new Link<ulong>(links.GetLink(linkIndex));
204             if (renderIndex)
205             {
206                 sb.Append(link.Index);
207                 sb.Append(':');
208             }
209             if (link.Source == link.Index)
210             {
211                 sb.Append(link.Index);
212             }
213             else
214             {
215                 var source = new Link<ulong>(links.GetLink(link.Source));
216                 if (isElement(source))
217                 {
218                     appendElement(sb, source);
219                 }
220                 else
221                 {
222                     links.AppendStructure(sb, visited, source.Index, isElement,
    ↳ appendElement, renderIndex);
223                 }
224             }
225             sb.Append(' ');
226             if (link.Target == link.Index)
227             {
228                 sb.Append(link.Index);
229             }
230             else
231             {
232                 var target = new Link<ulong>(links.GetLink(link.Target));
233                 if (isElement(target))
234                 {
235                     appendElement(sb, target);
236                 }
237                 else
238                 {
239                     links.AppendStructure(sb, visited, target.Index, isElement,
    ↳ appendElement, renderIndex);

```

```

240         }
241     }
242     sb.Append(' ');
243 }
244 else
245 {
246     if (renderDebug)
247     {
248         sb.Append('*');
249     }
250     sb.Append(linkIndex);
251 }
252 }
253 else
254 {
255     if (renderDebug)
256     {
257         sb.Append('~');
258     }
259     sb.Append(linkIndex);
260 }
261 }
262 }
263 }

```

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

```

1  using System;
2  using System.Linq;
3  using System.Collections.Generic;
4  using System.IO;
5  using System.Runtime.CompilerServices;
6  using System.Threading;
7  using System.Threading.Tasks;
8  using Platform.Disposables;
9  using Platform.Timestamps;
10 using Platform.Unsafe;
11 using Platform.IO;
12 using Platform.Data.Doublets.Decorators;
13 using Platform.Exceptions;
14
15 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
16
17 namespace Platform.Data.Doublets
18 {
19     /// <summary>
20     /// <para>
21     /// Represents the int 64 links transactions layer.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     /// <seealso cref="LinksDisposableDecoratorBase{ulong}" />
26     public class UInt64LinksTransactionsLayer : LinksDisposableDecoratorBase<ulong> //-V3073
27     {
28         /// <remarks>
29         /// Альтернативные варианты хранения трансформации (элемента транзакции):
30         ///
31         /// private enum TransitionType
32         /// {
33         ///     Creation,
34         ///     UpdateOf,
35         ///     UpdateTo,
36         ///     Deletion
37         /// }
38         ///
39         /// private struct Transition
40         /// {
41         ///     public ulong TransactionId;
42         ///     public UniqueTimestamp Timestamp;
43         ///     public TransactionItemType Type;
44         ///     public Link Source;
45         ///     public Link Linker;
46         ///     public Link Target;
47         /// }
48         ///
49         /// Или
50         ///
51         /// public struct TransitionHeader
52         /// {
53         ///     public ulong TransactionIdCombined;

```

```

54     /// public ulong TimestampCombined;
55     ///
56     /// public ulong TransactionId
57     /// {
58     ///     get
59     ///     {
60     ///         return (ulong) mask & TransactionIdCombined;
61     ///     }
62     /// }
63     ///
64     /// public UniqueTimestamp Timestamp
65     /// {
66     ///     get
67     ///     {
68     ///         return (UniqueTimestamp)mask & TransactionIdCombined;
69     ///     }
70     /// }
71     ///
72     /// public TransactionItemType Type
73     /// {
74     ///     get
75     ///     {
76     ///         // Использовать по одному биту из TransactionId и Timestamp,
77     ///         // для значения в 2 бита, которое представляет тип операции
78     ///         throw new NotImplementedException();
79     ///     }
80     /// }
81     /// }
82     ///
83     /// private struct Transition
84     /// {
85     ///     public TransitionHeader Header;
86     ///     public Link Source;
87     ///     public Link Linker;
88     ///     public Link Target;
89     /// }
90     ///
91     /// </remarks>
92     public struct Transition : IEquatable<Transition>
93     {
94         /// <summary>
95         /// <para>
96         /// The size.
97         /// </para>
98         /// <para></para>
99         /// </summary>
100        public static readonly long Size = Structure<Transition>.Size;
101
102        /// <summary>
103        /// <para>
104        /// The transaction id.
105        /// </para>
106        /// <para></para>
107        /// </summary>
108        public readonly ulong TransactionId;
109        /// <summary>
110        /// <para>
111        /// The before.
112        /// </para>
113        /// <para></para>
114        /// </summary>
115        public readonly Link<ulong> Before;
116        /// <summary>
117        /// <para>
118        /// The after.
119        /// </para>
120        /// <para></para>
121        /// </summary>
122        public readonly Link<ulong> After;
123        /// <summary>
124        /// <para>
125        /// The timestamp.
126        /// </para>
127        /// <para></para>
128        /// </summary>
129        public readonly Timestamp Timestamp;
130
131        /// <summary>

```

```

132     /// <para>
133     /// Initializes a new <see cref="Transition"/> instance.
134     /// </para>
135     /// <para></para>
136     /// </summary>
137     /// <param name="uniqueTimestampFactory">
138     /// <para>A unique timestamp factory.</para>
139     /// <para></para>
140     /// </param>
141     /// <param name="transactionId">
142     /// <para>A transaction id.</para>
143     /// <para></para>
144     /// </param>
145     /// <param name="before">
146     /// <para>A before.</para>
147     /// <para></para>
148     /// </param>
149     /// <param name="after">
150     /// <para>A after.</para>
151     /// <para></para>
152     /// </param>
153     [MethodImpl(MethodImplOptions.AggressiveInlining)]
154     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId, Link<ulong> before, Link<ulong> after)
155     {
156         TransactionId = transactionId;
157         Before = before;
158         After = after;
159         Timestamp = uniqueTimestampFactory.Create();
160     }
161
162     /// <summary>
163     /// <para>
164     /// Initializes a new <see cref="Transition"/> instance.
165     /// </para>
166     /// <para></para>
167     /// </summary>
168     /// <param name="uniqueTimestampFactory">
169     /// <para>A unique timestamp factory.</para>
170     /// <para></para>
171     /// </param>
172     /// <param name="transactionId">
173     /// <para>A transaction id.</para>
174     /// <para></para>
175     /// </param>
176     /// <param name="before">
177     /// <para>A before.</para>
178     /// <para></para>
179     /// </param>
180     [MethodImpl(MethodImplOptions.AggressiveInlining)]
181     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId, Link<ulong> before) : this(uniqueTimestampFactory, transactionId,
        ↳ before, default) { }
182
183     /// <summary>
184     /// <para>
185     /// Initializes a new <see cref="Transition"/> instance.
186     /// </para>
187     /// <para></para>
188     /// </summary>
189     /// <param name="uniqueTimestampFactory">
190     /// <para>A unique timestamp factory.</para>
191     /// <para></para>
192     /// </param>
193     /// <param name="transactionId">
194     /// <para>A transaction id.</para>
195     /// <para></para>
196     /// </param>
197     [MethodImpl(MethodImplOptions.AggressiveInlining)]
198     public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↳ transactionId) : this(uniqueTimestampFactory, transactionId, default, default) {
        ↳ }
199
200     /// <summary>
201     /// <para>
202     /// Returns the string.
203     /// </para>
204     /// <para></para>

```

```

205     /// </summary>
206     /// <returns>
207     /// <para>The string</para>
208     /// <para></para>
209     /// </returns>
210     [MethodImpl(MethodImplOptions.AggressiveInlining)]
211     public override string ToString() => $"{Timestamp} {TransactionId}: {Before} =>
        ↳ {After}";
212
213     /// <summary>
214     /// <para>
215     /// Determines whether this instance equals.
216     /// </para>
217     /// <para></para>
218     /// </summary>
219     /// <param name="obj">
220     /// <para>The obj.</para>
221     /// <para></para>
222     /// </param>
223     /// <returns>
224     /// <para>The bool</para>
225     /// <para></para>
226     /// </returns>
227     [MethodImpl(MethodImplOptions.AggressiveInlining)]
228     public override bool Equals(object obj) => obj is Transition transition ?
        ↳ Equals(transition) : false;
229
230     /// <summary>
231     /// <para>
232     /// Gets the hash code.
233     /// </para>
234     /// <para></para>
235     /// </summary>
236     /// <returns>
237     /// <para>The int</para>
238     /// <para></para>
239     /// </returns>
240     [MethodImpl(MethodImplOptions.AggressiveInlining)]
241     public override int GetHashCode() => (TransactionId, Before, After,
        ↳ Timestamp).GetHashCode();
242
243     /// <summary>
244     /// <para>
245     /// Determines whether this instance equals.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     /// <param name="other">
250     /// <para>The other.</para>
251     /// <para></para>
252     /// </param>
253     /// <returns>
254     /// <para>The bool</para>
255     /// <para></para>
256     /// </returns>
257     [MethodImpl(MethodImplOptions.AggressiveInlining)]
258     public bool Equals(Transition other) => TransactionId == other.TransactionId &&
        ↳ Before == other.Before && After == other.After && Timestamp == other.Timestamp;
259
260     [MethodImpl(MethodImplOptions.AggressiveInlining)]
261     public static bool operator ==(Transition left, Transition right) =>
        ↳ left.Equals(right);
262
263     [MethodImpl(MethodImplOptions.AggressiveInlining)]
264     public static bool operator !=(Transition left, Transition right) => !(left ==
        ↳ right);
265 }
266
267 /// <remarks>
268 /// Другие варианты реализации транзакций (атомарности):
269 /// 1. Разделение хранения значения связи ((Source Target) или (Source Linker
        ↳ 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)
        будут записаны в лог.
293 ///
294 ///
295 </remarks>
296 public class Transaction : DisposableBase
297 {
298     /// <summary>
299     /// <para>
300     /// The transitions.
301     /// </para>
302     /// <para></para>
303     /// </summary>
304     private readonly Queue<Transition> _transitions;
305     /// <summary>
306     /// <para>
307     /// The layer.
308     /// </para>
309     /// <para></para>
310     /// </summary>
311     private readonly UInt64LinksTransactionsLayer _layer;
312     /// <summary>
313     /// <para>
314     /// Gets or sets the is committed value.
315     /// </para>
316     /// <para></para>
317     /// </summary>
318     public bool IsCommitted { get; private set; }
319     /// <summary>
320     /// <para>
321     /// Gets or sets the is reverted value.
322     /// </para>
323     /// <para></para>
324     /// </summary>
325     public bool IsReverted { get; private set; }
326
327     /// <summary>
328     /// <para>
329     /// Initializes a new <see cref="Transaction"/> instance.
330     /// </para>
331     /// <para></para>
332     /// </summary>
333     /// <param name="layer">
334     /// <para>A layer.</para>
335     /// <para></para>
336     /// </param>
337     /// <exception cref="NotSupportedException">
338     /// <para>Nested transactions not supported.</para>
339     /// <para></para>
340     /// </exception>
341     [MethodImpl(MethodImplOptions.AggressiveInlining)]
342     public Transaction(UInt64LinksTransactionsLayer layer)
343     {
344         _layer = layer;
345         if (_layer._currentTransactionId != 0)
346         {
347             throw new NotSupportedException("Nested transactions not supported.");

```

```

348     }
349     IsCommitted = false;
350     IsReverted = false;
351     _transitions = new Queue<Transition>();
352     SetCurrentTransaction(layer, this);
353 }
354
355 /// <summary>
356 /// <para>
357 /// Commits this instance.
358 /// </para>
359 /// <para></para>
360 /// </summary>
361 [MethodImpl(MethodImplOptions.AggressiveInlining)]
362 public void Commit()
363 {
364     EnsureTransactionAllowsWriteOperations(this);
365     while (_transitions.Count > 0)
366     {
367         var transition = _transitions.Dequeue();
368         _layer._transitions.Enqueue(transition);
369     }
370     _layer._lastCommittedTransactionId = _layer._currentTransactionId;
371     IsCommitted = true;
372 }
373
374 /// <summary>
375 /// <para>
376 /// Reverts this instance.
377 /// </para>
378 /// <para></para>
379 /// </summary>
380 [MethodImpl(MethodImplOptions.AggressiveInlining)]
381 private void Revert()
382 {
383     EnsureTransactionAllowsWriteOperations(this);
384     var transitionsToRevert = new Transition[_transitions.Count];
385     _transitions.CopyTo(transitionsToRevert, 0);
386     for (var i = transitionsToRevert.Length - 1; i >= 0; i--)
387     {
388         _layer.RevertTransition(transitionsToRevert[i]);
389     }
390     IsReverted = true;
391 }
392
393 /// <summary>
394 /// <para>
395 /// Sets the current transaction using the specified layer.
396 /// </para>
397 /// <para></para>
398 /// </summary>
399 /// <param name="layer">
400 /// <para>The layer.</para>
401 /// <para></para>
402 /// </param>
403 /// <param name="transaction">
404 /// <para>The transaction.</para>
405 /// <para></para>
406 /// </param>
407 [MethodImpl(MethodImplOptions.AggressiveInlining)]
408 public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
409 ↪ Transaction transaction)
410 {
411     layer._currentTransactionId = layer._lastCommittedTransactionId + 1;
412     layer._currentTransactionTransitions = transaction._transitions;
413     layer._currentTransaction = transaction;
414 }
415
416 /// <summary>
417 /// <para>
418 /// Ensures the transaction allows write operations using the specified transaction.
419 /// </para>
420 /// <para></para>
421 /// </summary>
422 /// <param name="transaction">
423 /// <para>The transaction.</para>
424 /// <para></para>
425 /// </param>

```



```

425     /// <exception cref="InvalidOperationException">
426     /// <para>Transation is committed.</para>
427     /// <para></para>
428     /// </exception>
429     /// <exception cref="InvalidOperationException">
430     /// <para>Transation is reverted.</para>
431     /// <para></para>
432     /// </exception>
433     [MethodImpl(MethodImplOptions.AggressiveInlining)]
434     public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
435     {
436         if (transaction.IsReverted)
437         {
438             throw new InvalidOperationException("Transation is reverted.");
439         }
440         if (transaction.IsCommitted)
441         {
442             throw new InvalidOperationException("Transation is committed.");
443         }
444     }
445
446     /// <summary>
447     /// <para>
448     /// Disposes the manual.
449     /// </para>
450     /// <para></para>
451     /// </summary>
452     /// <param name="manual">
453     /// <para>The manual.</para>
454     /// <para></para>
455     /// </param>
456     /// <param name="wasDisposed">
457     /// <para>The was disposed.</para>
458     /// <para></para>
459     /// </param>
460     [MethodImpl(MethodImplOptions.AggressiveInlining)]
461     protected override void Dispose(bool manual, bool wasDisposed)
462     {
463         if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
464         {
465             if (!IsCommitted && !IsReverted)
466             {
467                 Revert();
468             }
469             _layer.ResetCurrentTransation();
470         }
471     }
472 }
473
474 /// <summary>
475 /// <para>
476 /// The from seconds.
477 /// </para>
478 /// <para></para>
479 /// </summary>
480 public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
481
482 /// <summary>
483 /// <para>
484 /// The log address.
485 /// </para>
486 /// <para></para>
487 /// </summary>
488 private readonly string _logAddress;
489
490 /// <summary>
491 /// <para>
492 /// The log.
493 /// </para>
494 /// <para></para>
495 /// </summary>
496 private readonly FileStream _log;
497
498 /// <summary>
499 /// <para>
500 /// The transitions.
501 /// </para>
502 /// <para></para>
503 /// </summary>
504 private readonly Queue<Transition> _transitions;

```

```

503     /// <summary>
504     /// <para>
505     /// The unique timestamp factory.
506     /// </para>
507     /// <para></para>
508     /// </summary>
509     private readonly UniqueTimestampFactory _uniqueTimestampFactory;
510     /// <summary>
511     /// <para>
512     /// The transitions pusher.
513     /// </para>
514     /// <para></para>
515     /// </summary>
516     private Task _transitionsPusher;
517     /// <summary>
518     /// <para>
519     /// The last committed transition.
520     /// </para>
521     /// <para></para>
522     /// </summary>
523     private Transition _lastCommittedTransition;
524     /// <summary>
525     /// <para>
526     /// The current transaction id.
527     /// </para>
528     /// <para></para>
529     /// </summary>
530     private ulong _currentTransactionId;
531     /// <summary>
532     /// <para>
533     /// The current transaction transitions.
534     /// </para>
535     /// <para></para>
536     /// </summary>
537     private Queue<Transition> _currentTransactionTransitions;
538     /// <summary>
539     /// <para>
540     /// The current transaction.
541     /// </para>
542     /// <para></para>
543     /// </summary>
544     private Transaction _currentTransaction;
545     /// <summary>
546     /// <para>
547     /// The last committed transaction id.
548     /// </para>
549     /// <para></para>
550     /// </summary>
551     private ulong _lastCommittedTransactionId;
552
553     /// <summary>
554     /// <para>
555     /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.
556     /// </para>
557     /// <para></para>
558     /// </summary>
559     /// <param name="links">
560     /// <para>A links.</para>
561     /// <para></para>
562     /// </param>
563     /// <param name="logAddress">
564     /// <para>A log address.</para>
565     /// <para></para>
566     /// </param>
567     /// <exception cref="ArgumentNullException">
568     /// <para></para>
569     /// <para></para>
570     /// </exception>
571     /// <exception cref="NotSupportedException">
572     /// <para>Database is damaged, autorecovery is not supported yet.</para>
573     /// <para></para>
574     /// </exception>
575     [MethodImpl(MethodImplOptions.AggressiveInlining)]
576     public UInt64LinksTransactionsLayer(ILinks<ulong> links, string logAddress)
577         : base(links)
578     {
579         if (string.IsNullOrEmpty(logAddress))
580         {

```

```

581         throw new ArgumentNullException(nameof(logAddress));
582     }
583     // В первой строке файла хранится последняя законченная транзакция.
584     // При запуске это используется для проверки удачного закрытия файла лога.
585     // In the first line of the file the last committed transaction is stored.
586     // On startup, this is used to check that the log file is successfully closed.
587     var lastCommittedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
588     var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
589     if (!lastCommittedTransition.Equals(lastWrittenTransition))
590     {
591         Dispose();
592         throw new NotSupportedException("Database is damaged, autorecovery is not
593             ↳ supported yet.");
594     }
595     if (lastCommittedTransition == default)
596     {
597         FileHelpers.WriteFirst(logAddress, lastCommittedTransition);
598     }
599     _lastCommittedTransition = lastCommittedTransition;
600     // TODO: Think about a better way to calculate or store this value
601     var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
602     _lastCommittedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
603         ↳ x.TransactionId) : 0;
604     _uniqueTimestampFactory = new UniqueTimestampFactory();
605     _logAddress = logAddress;
606     _log = FileHelpers.Append(logAddress);
607     _transitions = new Queue<Transition>();
608     _transitionsPusher = new Task(TransitionsPusher);
609     _transitionsPusher.Start();
610 }
611
612 /// <summary>
613 /// <para>
614 /// Gets the link value using the specified link.
615 /// </para>
616 /// </summary>
617 /// <param name="link">
618 /// <para>The link.</para>
619 /// </param>
620 /// <returns>
621 /// <para>A list of ulong</para>
622 /// </returns>
623 [MethodImpl(MethodImplOptions.AggressiveInlining)]
624 public IList<ulong> GetLinkValue(ulong link) => _links.GetLink(link);
625
626 /// <summary>
627 /// <para>
628 /// Creates the restrictions.
629 /// </para>
630 /// </summary>
631 /// <param name="restrictions">
632 /// <para>The restrictions.</para>
633 /// </param>
634 /// <returns>
635 /// <para>The created link index.</para>
636 /// </returns>
637 [MethodImpl(MethodImplOptions.AggressiveInlining)]
638 public override ulong Create(IList<ulong> restrictions)
639 {
640     var createdLinkIndex = _links.Create();
641     var createdLink = new Link<ulong>(_links.GetLink(createdLinkIndex));
642     CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
643         ↳ default, createdLink));
644     return createdLinkIndex;
645 }
646
647 /// <summary>
648 /// <para>
649 /// Updates the restrictions.
650 /// </para>
651 /// </summary>
652

```

```

656    /// <param name="restrictions">
657    /// <para>The restrictions.</para>
658    /// <para></para>
659    /// </param>
660    /// <param name="substitution">
661    /// <para>The substitution.</para>
662    /// <para></para>
663    /// </param>
664    /// <returns>
665    /// <para>The link index.</para>
666    /// <para></para>
667    /// </returns>
668    [MethodImpl(MethodImplOptions.AggressiveInlining)]
669    public override ulong Update(IList<ulong> restrictions, IList<ulong> substitution)
670    {
671        var linkIndex = restrictions[_constants.IndexPart];
672        var beforeLink = new Link<ulong>(_links.GetLink(linkIndex));
673        linkIndex = _links.Update(restrictions, substitution);
674        var afterLink = new Link<ulong>(_links.GetLink(linkIndex));
675        CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
676            ↪ beforeLink, afterLink));
677        return linkIndex;
678    }
679    /// <summary>
680    /// <para>
681    /// Deletes the restrictions.
682    /// </para>
683    /// <para></para>
684    /// </summary>
685    /// <param name="restrictions">
686    /// <para>The restrictions.</para>
687    /// <para></para>
688    /// </param>
689    [MethodImpl(MethodImplOptions.AggressiveInlining)]
690    public override void Delete(IList<ulong> restrictions)
691    {
692        var link = restrictions[_constants.IndexPart];
693        var deletedLink = new Link<ulong>(_links.GetLink(link));
694        _links.Delete(link);
695        CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
696            ↪ deletedLink, default));
697    }
698    /// <summary>
699    /// <para>
700    /// Gets the current transitions.
701    /// </para>
702    /// <para></para>
703    /// </summary>
704    /// <returns>
705    /// <para>A queue of transition</para>
706    /// <para></para>
707    /// </returns>
708    [MethodImpl(MethodImplOptions.AggressiveInlining)]
709    private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
710        ↪ _transitions;
711    /// <summary>
712    /// <para>
713    /// Commits the transition using the specified transition.
714    /// </para>
715    /// <para></para>
716    /// </summary>
717    /// <param name="transition">
718    /// <para>The transition.</para>
719    /// <para></para>
720    /// </param>
721    [MethodImpl(MethodImplOptions.AggressiveInlining)]
722    private void CommitTransition(Transition transition)
723    {
724        if (_currentTransaction != null)
725        {
726            Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
727        }
728        var transitions = GetCurrentTransitions();
729        transitions.Enqueue(transition);
730    }

```

```

731 /// <summary>
732 /// <para>
733 /// Reverts the transition using the specified transition.
734 /// </para>
735 /// <para></para>
736 /// </summary>
737 /// <param name="transition">
738 /// <para>The transition.</para>
739 /// <para></para>
740 /// </param>
741 [MethodImpl(MethodImplOptions.AggressiveInlining)]
742 private void RevertTransition(Transition transition)
743 {
744     if (transition.After.IsNull() // Revert Deletion with Creation
745     {
746         _links.Create();
747     }
748     else if (transition.Before.IsNull() // Revert Creation with Deletion
749     {
750         _links.Delete(transition.After.Index);
751     }
752     else // Revert Update
753     {
754         _links.Update(new[] { transition.After.Index, transition.Before.Source,
755             ↪ transition.Before.Target });
756     }
757 }
758
759 /// <summary>
760 /// <para>
761 /// Resets the current transation.
762 /// </para>
763 /// <para></para>
764 /// </summary>
765 [MethodImpl(MethodImplOptions.AggressiveInlining)]
766 private void ResetCurrentTransation()
767 {
768     _currentTransactionId = 0;
769     _currentTransactionTransitions = null;
770     _currentTransaction = null;
771 }
772
773 /// <summary>
774 /// <para>
775 /// Pushes the transitions.
776 /// </para>
777 /// <para></para>
778 /// </summary>
779 [MethodImpl(MethodImplOptions.AggressiveInlining)]
780 private void PushTransitions()
781 {
782     if (_log == null || _transitions == null)
783     {
784         return;
785     }
786     for (var i = 0; i < _transitions.Count; i++)
787     {
788         var transition = _transitions.Dequeue();
789
790         _log.Write(transition);
791         _lastCommittedTransition = transition;
792     }
793 }
794
795 /// <summary>
796 /// <para>
797 /// Transitioneses the pusher.
798 /// </para>
799 /// <para></para>
800 /// </summary>
801 [MethodImpl(MethodImplOptions.AggressiveInlining)]
802 private void TransitionsPusher()
803 {
804     while (!Disposable.IsDisposed && _transitionsPusher != null)
805     {
806         Thread.Sleep(DefaultPushDelay);
807         PushTransitions();

```

```

808     }
809 }
810
811 /// <summary>
812 /// <para>
813 /// Begins the transaction.
814 /// </para>
815 /// <para></para>
816 /// </summary>
817 /// <returns>
818 /// <para>The transaction</para>
819 /// <para></para>
820 /// </returns>
821 [MethodImpl(MethodImplOptions.AggressiveInlining)]
822 public Transaction BeginTransaction() => new Transaction(this);
823
824 /// <summary>
825 /// <para>
826 /// Disposes the transitions.
827 /// </para>
828 /// <para></para>
829 /// </summary>
830 [MethodImpl(MethodImplOptions.AggressiveInlining)]
831 private void DisposeTransitions()
832 {
833     try
834     {
835         var pusher = _transitionsPusher;
836         if (pusher != null)
837         {
838             _transitionsPusher = null;
839             pusher.Wait();
840         }
841         if (_transitions != null)
842         {
843             PushTransitions();
844         }
845         _log.DisposeIfPossible();
846         FileHelpers.WriteFirst(_logAddress, _lastCommittedTransition);
847     }
848     catch (Exception ex)
849     {
850         ex.Ignore();
851     }
852 }
853
854 #region DisposalBase
855
856 /// <summary>
857 /// <para>
858 /// Disposes the manual.
859 /// </para>
860 /// <para></para>
861 /// </summary>
862 /// <param name="manual">
863 /// <para>The manual.</para>
864 /// <para></para>
865 /// </param>
866 /// <param name="wasDisposed">
867 /// <para>The was disposed.</para>
868 /// <para></para>
869 /// </param>
870 [MethodImpl(MethodImplOptions.AggressiveInlining)]
871 protected override void Dispose(bool manual, bool wasDisposed)
872 {
873     if (!wasDisposed)
874     {
875         DisposeTransitions();
876     }
877     base.Dispose(manual, wasDisposed);
878 }
879
880 #endregion
881 }
882 }

```

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

```

1 using System;
2 using Xunit;

```

```

3 using Platform.Reflection;
4 using Platform.Memory;
5 using Platform.Scopes;
6 using Platform.Data.Doublets.Memory.United.Generic;
7
8 namespace Platform.Data.Doublets.Tests
9 {
10     /// <summary>
11     /// <para>
12     /// Represents the generic links tests.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public unsafe static class GenericLinksTests
17     {
18         /// <summary>
19         /// <para>
20         /// Tests that crud test.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         [Fact]
25         public static void CRUDTest()
26         {
27             Using<byte>(links => links.TestCRUDOperations());
28             Using<ushort>(links => links.TestCRUDOperations());
29             Using<uint>(links => links.TestCRUDOperations());
30             Using<ulong>(links => links.TestCRUDOperations());
31         }
32
33         /// <summary>
34         /// <para>
35         /// Tests that raw numbers crud test.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         [Fact]
40         public static void RawNumbersCRUDTest()
41         {
42             Using<byte>(links => links.TestRawNumbersCRUDOperations());
43             Using<ushort>(links => links.TestRawNumbersCRUDOperations());
44             Using<uint>(links => links.TestRawNumbersCRUDOperations());
45             Using<ulong>(links => links.TestRawNumbersCRUDOperations());
46         }
47
48         /// <summary>
49         /// <para>
50         /// Tests that multiple random creations and deletions test.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         [Fact]
55         public static void MultipleRandomCreationsAndDeletionsTest()
56         {
57             Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
58                 ↳ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
59                 ↳ implementation of tree cuts out 5 bits from the address space.
60             Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
61                 ↳ stMultipleRandomCreationsAndDeletions(100));
62             Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
63                 ↳ MultipleRandomCreationsAndDeletions(100));
64             Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
65                 ↳ tMultipleRandomCreationsAndDeletions(100));
66         }
67
68         /// <summary>
69         /// <para>
70         /// Usings the action.
71         /// </para>
72         /// <para></para>
73         /// </summary>
74         /// <typeparam name="TLink">
75         /// <para>The link.</para>
76         /// <para></para>
77         /// </typeparam>
78         /// <param name="action">
79         /// <para>The action.</para>
80         /// <para></para>

```

```

76     /// </param>
77     private static void Using<TLink>(Action<ILinks<TLink>> action)
78     {
79         using (var scope = new Scope<Types<HeapResizableDirectMemory,
80             ↳ UnitedMemoryLinks<TLink>>>())
81         {
82             action(scope.Use<ILinks<TLink>>());
83         }
84     }
85 }

```

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

```

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

```

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

```

1  using System.IO;
2  using Xunit;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.United.Specific;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the resizable direct memory links tests.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public static class ResizableDirectMemoryLinksTests
16     {
17         /// <summary>
18         /// <para>
19         /// The instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         private static readonly LinksConstants<ulong> _constants =
24             ↳ Default<LinksConstants<ulong>>.Instance;
25
26         /// <summary>
27         /// <para>
28         /// Tests that basic file mapped memory test.
29         /// </para>
30         /// <para></para>

```



```

30     /// </summary>
31     [Fact]
32     public static void BasicFileMappedMemoryTest()
33     {
34         var tempFilename = Path.GetTempFileName();
35         using (var memoryAdapter = new UInt64UnitedMemoryLinks(tempFilename))
36         {
37             memoryAdapter.TestBasicMemoryOperations();
38         }
39         File.Delete(tempFilename);
40     }
41
42     /// <summary>
43     /// <para>
44     /// Tests that basic heap memory test.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     [Fact]
49     public static void BasicHeapMemoryTest()
50     {
51         using (var memory = new
52             ↳ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
53         using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
54             ↳ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
55         {
56             memoryAdapter.TestBasicMemoryOperations();
57         }
58     }
59
60     /// <summary>
61     /// <para>
62     /// Tests the basic memory operations using the specified memory adapter.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="memoryAdapter">
67     /// <para>The memory adapter.</para>
68     /// <para></para>
69     /// </param>
70     private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
71     {
72         var link = memoryAdapter.Create();
73         memoryAdapter.Delete(link);
74     }
75
76     /// <summary>
77     /// <para>
78     /// Tests that nonexistent references heap memory test.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     [Fact]
83     public static void NonexistentReferencesHeapMemoryTest()
84     {
85         using (var memory = new
86             ↳ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
87         using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
88             ↳ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
89         {
90             memoryAdapter.TestNonexistentReferences();
91         }
92     }
93
94     /// <summary>
95     /// <para>
96     /// Tests the nonexistent references using the specified memory adapter.
97     /// </para>
98     /// <para></para>
99     /// </summary>
100    /// <param name="memoryAdapter">
101    /// <para>The memory adapter.</para>
102    /// <para></para>
103    /// </param>
104    private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
105    {
106        var link = memoryAdapter.Create();
107        memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);

```

```

104     var resultLink = _constants.Null;
105     memoryAdapter.Each(foundLink =>
106     {
107         resultLink = foundLink[_constants.IndexPart];
108         return _constants.Break;
109     }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
110     Assert.True(resultLink == link);
111     Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
112     memoryAdapter.Delete(link);
113 }
114 }
115 }

```

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

```

1  using Xunit;
2  using Platform.Scopes;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Reflection;
6  using Platform.Data.Doublets.Memory.United.Generic;
7  using Platform.Data.Doublets.Memory.United.Specific;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the scope tests.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class ScopeTests
18     {
19         /// <summary>
20         /// <para>
21         /// Tests that single dependency test.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         [Fact]
26         public static void SingleDependencyTest()
27         {
28             using (var scope = new Scope())
29             {
30                 scope.IncludeAssemblyOf<IMemory>();
31                 var instance = scope.Use<IDirectMemory>();
32                 Assert.IsType<HeapResizableDirectMemory>(instance);
33             }
34         }
35
36         /// <summary>
37         /// <para>
38         /// Tests that cascade dependency test.
39         /// </para>
40         /// <para></para>
41         /// </summary>
42         [Fact]
43         public static void CascadeDependencyTest()
44         {
45             using (var scope = new Scope())
46             {
47                 scope.Include<TemporaryFileMappedResizableDirectMemory>();
48                 scope.Include<UInt64UnitedMemoryLinks>();
49                 var instance = scope.Use<ILinks<ulong>>();
50                 Assert.IsType<UInt64UnitedMemoryLinks>(instance);
51             }
52         }
53
54         /// <summary>
55         /// <para>
56         /// Tests that full auto resolution test.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         [Fact(Skip = "Would be fixed later.")]
61         public static void FullAutoResolutionTest()
62         {
63             using (var scope = new Scope(autoInclude: true, autoExplore: true))
64             {
65                 var instance = scope.Use<UInt64Links>();

```

```

66         Assert.IsType<UInt64Links>(instance);
67     }
68 }
69
70 /// <summary>
71 /// <para>
72 /// Tests that type parameters test.
73 /// </para>
74 /// <para></para>
75 /// </summary>
76 [Fact]
77 public static void TypeParametersTest()
78 {
79     using (var scope = new Scope<Types<HeapResizableDirectMemory,
80         ↪ UnitedMemoryLinks<ulong>>>())
81     {
82         var links = scope.Use<ILinks<ulong>>>();
83         Assert.IsType<UnitedMemoryLinks<ulong>>>(links);
84     }
85 }
86 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Generic;
5  using Platform.Data.Doublets.Memory;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the split memory generic links tests.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public unsafe static class SplitMemoryGenericLinksTests
16     {
17         /// <summary>
18         /// <para>
19         /// Tests that crud test.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         [Fact]
24         public static void CRUDTest()
25         {
26             Using<byte>(links => links.TestCRUDOperations());
27             Using<ushort>(links => links.TestCRUDOperations());
28             Using<uint>(links => links.TestCRUDOperations());
29             Using<ulong>(links => links.TestCRUDOperations());
30         }
31
32         /// <summary>
33         /// <para>
34         /// Tests that raw numbers crud test.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         [Fact]
39         public static void RawNumbersCRUDTest()
40         {
41             UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
42             UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
43             UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
44             UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
45         }
46
47         /// <summary>
48         /// <para>
49         /// Tests that multiple random creations and deletions test.
50         /// </para>
51         /// <para></para>
52         /// </summary>
53         [Fact]
54         public static void MultipleRandomCreationsAndDeletionsTest()
55         {

```

```

56     Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
    ↪ MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
    ↪ implementation of tree cuts out 5 bits from the address space.
57     Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
    ↪ stMultipleRandomCreationsAndDeletions(100));
58     Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
    ↪ MultipleRandomCreationsAndDeletions(100));
59     Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
    ↪ tMultipleRandomCreationsAndDeletions(100));
60 }
61
62 /// <summary>
63 /// <para>
64 /// Usings the action.
65 /// </para>
66 /// <para></para>
67 /// </summary>
68 /// <typeparam name="TLink">
69 /// <para>The link.</para>
70 /// <para></para>
71 /// </typeparam>
72 /// <param name="action">
73 /// <para>The action.</para>
74 /// <para></para>
75 /// </param>
76 private static void Using<TLink>(Action<ILinks<TLink>> action)
77 {
78     using (var dataMemory = new HeapResizableDirectMemory())
79     using (var indexMemory = new HeapResizableDirectMemory())
80     using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory))
81     {
82         action(memory);
83     }
84 }
85
86 /// <summary>
87 /// <para>
88 /// Usings the with external references using the specified action.
89 /// </para>
90 /// <para></para>
91 /// </summary>
92 /// <typeparam name="TLink">
93 /// <para>The link.</para>
94 /// <para></para>
95 /// </typeparam>
96 /// <param name="action">
97 /// <para>The action.</para>
98 /// <para></para>
99 /// </param>
100 private static void UsingWithExternalReferences<TLink>(Action<ILinks<TLink>> action)
101 {
102     var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
103     using (var dataMemory = new HeapResizableDirectMemory())
104     using (var indexMemory = new HeapResizableDirectMemory())
105     using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory,
    ↪ SplitMemoryLinks<TLink>.DefaultLinksSizeStep, constants))
106     {
107         action(memory);
108     }
109 }
110 }
111 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLink = System.UInt32;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the split memory int 32 links tests.
12     /// </para>
13     /// <para></para>
14     /// </summary>

```

```

15 public unsafe static class SplitMemoryUInt32LinksTests
16 {
17     /// <summary>
18     /// <para>
19     /// Tests that crud test.
20     /// </para>
21     /// <para></para>
22     /// </summary>
23     [Fact]
24     public static void CRUDTest()
25     {
26         Using(links => links.TestCRUDOperations());
27     }
28
29     /// <summary>
30     /// <para>
31     /// Tests that raw numbers crud test.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     [Fact]
36     public static void RawNumbersCRUDTest()
37     {
38         UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
39     }
40
41     /// <summary>
42     /// <para>
43     /// Tests that multiple random creations and deletions test.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     [Fact]
48     public static void MultipleRandomCreationsAndDeletionsTest()
49     {
50         Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
51             ↪ leRandomCreationsAndDeletions(500));
52     }
53
54     /// <summary>
55     /// <para>
56     /// Usings the action.
57     /// </para>
58     /// <para></para>
59     /// </summary>
60     /// <param name="action">
61     /// <para>The action.</para>
62     /// <para></para>
63     /// </param>
64     private static void Using(Action<ILinks<TLink>> action)
65     {
66         using (var dataMemory = new HeapResizableDirectMemory())
67         using (var indexMemory = new HeapResizableDirectMemory())
68         using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory))
69         {
70             action(memory);
71         }
72     }
73
74     /// <summary>
75     /// <para>
76     /// Usings the with external references using the specified action.
77     /// </para>
78     /// <para></para>
79     /// </summary>
80     /// <param name="action">
81     /// <para>The action.</para>
82     /// <para></para>
83     /// </param>
84     private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
85     {
86         var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
87         using (var dataMemory = new HeapResizableDirectMemory())
88         using (var indexMemory = new HeapResizableDirectMemory())
89         using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory,
90             ↪ UInt32SplitMemoryLinks.DefaultLinksSizeStep, constants))
91         {
92             action(memory);
93         }
94     }
95 }

```

```

91     }
92 }
93 }
94 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Memory.Split.Specific;
5  using TLink = System.UInt64;
6
7  namespace Platform.Data.Doublets.Tests
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the split memory int 64 links tests.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     public unsafe static class SplitMemoryUInt64LinksTests
16     {
17         /// <summary>
18         /// <para>
19         /// Tests that crud test.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         [Fact]
24         public static void CRUDTest()
25         {
26             Using(links => links.TestCRUDOperations());
27         }
28
29         /// <summary>
30         /// <para>
31         /// Tests that raw numbers crud test.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         [Fact]
36         public static void RawNumbersCRUDTest()
37         {
38             UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
39         }
40
41         /// <summary>
42         /// <para>
43         /// Tests that multiple random creations and deletions test.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         [Fact]
48         public static void MultipleRandomCreationsAndDeletionsTest()
49         {
50             Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(500));
51         }
52
53         /// <summary>
54         /// <para>
55         /// Uses the action.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         /// <param name="action">
60         /// <para>The action.</para>
61         /// <para></para>
62         /// </param>
63         private static void Using(Action<ILinks<TLink>> action)
64         {
65             using (var dataMemory = new HeapResizableDirectMemory())
66             using (var indexMemory = new HeapResizableDirectMemory())
67             using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory))
68             {
69                 action(memory);
70             }
71         }
72     }

```

```

73     /// <summary>
74     /// <para>
75     /// Uses the with external references using the specified action.
76     /// </para>
77     /// <para></para>
78     /// </summary>
79     /// <param name="action">
80     /// <para>The action.</para>
81     /// <para></para>
82     /// </param>
83     private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
84     {
85         var constants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
86         using (var dataMemory = new HeapResizableDirectMemory())
87         using (var indexMemory = new HeapResizableDirectMemory())
88         using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
89             ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, constants))
90         {
91             action(memory);
92         }
93     }
94 }

```

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

```

1  using System.Collections.Generic;
2  using Xunit;
3  using Platform.Ranges;
4  using Platform.Numbers;
5  using Platform.Random;
6  using Platform.Setters;
7  using Platform.Converters;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the test extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class TestExtensions
18     {
19         /// <summary>
20         /// <para>
21         /// Tests the crud operations using the specified links.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <typeparam name="T">
26         /// <para>The .</para>
27         /// <para></para>
28         /// </typeparam>
29         /// <param name="links">
30         /// <para>The links.</para>
31         /// <para></para>
32         /// </param>
33         public static void TestCRUDOperations<T>(this ILinks<T> links)
34         {
35             var constants = links.Constants;
36
37             var equalityComparer = EqualityComparer<T>.Default;
38
39             var zero = default(T);
40             var one = Arithmetic.Increment(zero);
41
42             // Create Link
43             Assert.True(equalityComparer.Equals(links.Count(), zero));
44
45             var setter = new Setter<T>(constants.Null);
46             links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
47
48             Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
49
50             var linkAddress = links.Create();
51
52             var link = new Link<T>(links.GetLink(linkAddress));
53
54             Assert.True(link.Count == 3);

```

```

55     Assert.True(equalityComparer.Equals(link.Index, linkAddress));
56     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
57     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
58
59     Assert.True(equalityComparer.Equals(links.Count(), one));
60
61     // Get first link
62     setter = new Setter<T>(constants.Null);
63     links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
64
65     Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
66
67     // Update link to reference itself
68     links.Update(linkAddress, linkAddress, linkAddress);
69
70     link = new Link<T>(links.GetLink(linkAddress));
71
72     Assert.True(equalityComparer.Equals(link.Source, linkAddress));
73     Assert.True(equalityComparer.Equals(link.Target, linkAddress));
74
75     // Update link to reference null (prepare for delete)
76     var updated = links.Update(linkAddress, constants.Null, constants.Null);
77
78     Assert.True(equalityComparer.Equals(updated, linkAddress));
79
80     link = new Link<T>(links.GetLink(linkAddress));
81
82     Assert.True(equalityComparer.Equals(link.Source, constants.Null));
83     Assert.True(equalityComparer.Equals(link.Target, constants.Null));
84
85     // Delete link
86     links.Delete(linkAddress);
87
88     Assert.True(equalityComparer.Equals(links.Count(), zero));
89
90     setter = new Setter<T>(constants.Null);
91     links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
92
93     Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
94 }
95
96 /// <summary>
97 /// <para>
98 /// Tests the raw numbers crud operations using the specified links.
99 /// </para>
100 /// <para></para>
101 /// </summary>
102 /// <typeparam name="T">
103 /// <para>The .</para>
104 /// <para></para>
105 /// </typeparam>
106 /// <param name="links">
107 /// <para>The links.</para>
108 /// <para></para>
109 /// </param>
110 public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
111 {
112     // Constants
113     var constants = links.Constants;
114     var equalityComparer = EqualityComparer<T>.Default;
115
116     var zero = default(T);
117     var one = Arithmetic.Increment(zero);
118     var two = Arithmetic.Increment(one);
119
120     var h106E = new Hybrid<T>(106L, isExternal: true);
121     var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
122     var h108E = new Hybrid<T>(-108L);
123
124     Assert.Equal(106L, h106E.AbsoluteValue);
125     Assert.Equal(107L, h107E.AbsoluteValue);
126     Assert.Equal(108L, h108E.AbsoluteValue);
127
128     // Create Link (External -> External)
129     var linkAddress1 = links.Create();
130
131     links.Update(linkAddress1, h106E, h108E);
132
133     var link1 = new Link<T>(links.GetLink(linkAddress1));
134

```



```

135 Assert.True(equalityComparer.Equals(link1.Source, h106E));
136 Assert.True(equalityComparer.Equals(link1.Target, h108E));
137
138 // Create Link (Internal -> External)
139 var linkAddress2 = links.Create();
140
141 links.Update(linkAddress2, linkAddress1, h108E);
142
143 var link2 = new Link<T>(links.GetLink(linkAddress2));
144
145 Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
146 Assert.True(equalityComparer.Equals(link2.Target, h108E));
147
148 // Create Link (Internal -> Internal)
149 var linkAddress3 = links.Create();
150
151 links.Update(linkAddress3, linkAddress1, linkAddress2);
152
153 var link3 = new Link<T>(links.GetLink(linkAddress3));
154
155 Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
156 Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));
157
158 // Search for created link
159 var setter1 = new Setter<T>(constants.Null);
160 links.Each(h106E, h108E, setter1.SetAndReturnFalse);
161
162 Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));
163
164 // Search for nonexistent link
165 var setter2 = new Setter<T>(constants.Null);
166 links.Each(h106E, h107E, setter2.SetAndReturnFalse);
167
168 Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));
169
170 // Update link to reference null (prepare for delete)
171 var updated = links.Update(linkAddress3, constants.Null, constants.Null);
172
173 Assert.True(equalityComparer.Equals(updated, linkAddress3));
174
175 link3 = new Link<T>(links.GetLink(linkAddress3));
176
177 Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
178 Assert.True(equalityComparer.Equals(link3.Target, constants.Null));
179
180 // Delete link
181 links.Delete(linkAddress3);
182
183 Assert.True(equalityComparer.Equals(links.Count(), two));
184
185 var setter3 = new Setter<T>(constants.Null);
186 links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);
187
188 Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
189 }
190
191 /// <summary>
192 /// <para>
193 /// Tests the multiple random creations and deletions using the specified links.
194 /// </para>
195 /// </summary>
196 /// <typeparam name="TLink">
197 /// <para>The link.</para>
198 /// </typeparam>
199 /// <param name="links">
200 /// <para>The links.</para>
201 /// </param>
202 /// <param name="maximumOperationsPerCycle">
203 /// <para>The maximum operations per cycle.</para>
204 /// </param>
205 public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
206 ↪ links, int maximumOperationsPerCycle)
207 {
208     var comparer = Comparer<TLink>.Default;
209     var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
210     var uint64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;

```

```

214     for (var N = 1; N < maximumOperationsPerCycle; N++)
215     {
216         var random = new System.Random(N);
217         var created = 0UL;
218         var deleted = 0UL;
219         for (var i = 0; i < N; i++)
220         {
221             var linksCount = addressToUInt64Converter.Convert(links.Count());
222             var createPoint = random.NextBoolean();
223             if (linksCount >= 2 && createPoint)
224             {
225                 var linksAddressRange = new Range<ulong>(1, linksCount);
226                 TLink source = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
                ↪ ddressRange));
227                 TLink target = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
                ↪ ddressRange));
228                 ↪ //-V3086
229                 var resultLink = links.GetOrCreate(source, target);
230                 if (comparer.Compare(resultLink,
                ↪ uInt64ToAddressConverter.Convert(linksCount)) > 0)
231                 {
232                     created++;
233                 }
234             }
235             else
236             {
237                 links.Create();
238                 created++;
239             }
240             Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
241             for (var i = 0; i < N; i++)
242             {
243                 TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
244                 if (links.Exists(link))
245                 {
246                     links.Delete(link);
247                     deleted++;
248                 }
249             }
250             Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
251         }
252     }
253 }
254 }

```

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

```

1  using Platform.Data.Doublets.Memory;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Numbers.Raw;
4  using Platform.Memory;
5  using Platform.Numbers;
6  using Xunit;
7  using Xunit.Abstractions;
8  using TLink = System.UInt64;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the uint 64 links extensions tests.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     public class UInt64LinksExtensionsTests
19     {
20         /// <summary>
21         /// <para>
22         /// Creates the links.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <returns>
27         /// <para>A links of t link</para>
28         /// <para></para>
29         /// </returns>
30         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
            ↪ Platform.IO.TemporaryFile());
31

```

```

32     /// <summary>
33     /// <para>
34     /// Creates the links using the specified data db filename.
35     /// </para>
36     /// <para></para>
37     /// </summary>
38     /// <typeparam name="TLink">
39     /// <para>The link.</para>
40     /// <para></para>
41     /// </typeparam>
42     /// <param name="dataDBFilename">
43     /// <para>The data db filename.</para>
44     /// <para></para>
45     /// </param>
46     /// <returns>
47     /// <para>A links of t link</para>
48     /// <para></para>
49     /// </returns>
50     public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
51     {
52         var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
53             ↪ true);
54         return new UnitedMemoryLinks<TLink>(new
55             ↪ FileMappedResizableDirectMemory(dataDBFilename),
56             ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
57             ↪ IndexTreeType.Default);
58     }
59     /// <summary>
60     /// <para>
61     /// Tests that format structure with external reference test.
62     /// </para>
63     /// <para></para>
64     /// </summary>
65     [Fact]
66     public void FormatStructureWithExternalReferenceTest()
67     {
68         ILinks<TLink> links = CreateLinks();
69         TLink zero = default;
70         var one = Arithmetic.Increment(zero);
71         var markerIndex = one;
72         var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
73         var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
74             ↪ markerIndex));
75         AddressToRawNumberConverter<TLink> addressToNumberConverter = new();
76         var numberAddress = addressToNumberConverter.Convert(1);
77         var numberLink = links.GetOrCreate(numberMarker, numberAddress);
78         var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
79             ↪ true);
80         Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
81     }
82 }
83 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLink = System.UInt32;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the united memory int 32 links tests.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public unsafe static class UnitedMemoryUInt32LinksTests
18     {
19         /// <summary>
20         /// <para>
21         /// Tests that crud test.
22         /// </para>
23         /// <para></para>
24         /// </summary>

```

```

25     [Fact]
26     public static void CRUDTest()
27     {
28         Using(links => links.TestCRUDOperations());
29     }
30
31     /// <summary>
32     /// <para>
33     /// Tests that raw numbers crud test.
34     /// </para>
35     /// <para></para>
36     /// </summary>
37     [Fact]
38     public static void RawNumbersCRUDTest()
39     {
40         Using(links => links.TestRawNumbersCRUDOperations());
41     }
42
43     /// <summary>
44     /// <para>
45     /// Tests that multiple random creations and deletions test.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     [Fact]
50     public static void MultipleRandomCreationsAndDeletionsTest()
51     {
52         Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
53     }
54
55     /// <summary>
56     /// <para>
57     /// Usings the action.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="action">
62     /// <para>The action.</para>
63     /// <para></para>
64     /// </param>
65     private static void Using(Action<ILinks<TLink>> action)
66     {
67         using (var scope = new Scope<Types<HeapResizableDirectMemory,
68             ↳ UInt32UnitedMemoryLinks>>())
69         {
70             action(scope.Use<ILinks<TLink>>());
71         }
72     }
73 }

```

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

```

1  using System;
2  using Xunit;
3  using Platform.Reflection;
4  using Platform.Memory;
5  using Platform.Scopes;
6  using Platform.Data.Doublets.Memory.United.Specific;
7  using TLink = System.UInt64;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the united memory int 64 links tests.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public unsafe static class UnitedMemoryUInt64LinksTests
18     {
19         /// <summary>
20         /// <para>
21         /// Tests that crud test.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         [Fact]
26         public static void CRUDTest()

```

```

27     {
28         Using(links => links.TestCRUDOperations());
29     }
30
31     /// <summary>
32     /// <para>
33     /// Tests that raw numbers crud test.
34     /// </para>
35     /// <para></para>
36     /// </summary>
37     [Fact]
38     public static void RawNumbersCRUDTest()
39     {
40         Using(links => links.TestRawNumbersCRUDOperations());
41     }
42
43     /// <summary>
44     /// <para>
45     /// Tests that multiple random creations and deletions test.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     [Fact]
50     public static void MultipleRandomCreationsAndDeletionsTest()
51     {
52         Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
53     }
54
55     /// <summary>
56     /// <para>
57     /// Usings the action.
58     /// </para>
59     /// <para></para>
60     /// </summary>
61     /// <param name="action">
62     /// <para>The action.</para>
63     /// <para></para>
64     /// </param>
65     private static void Using(Action<ILinks<TLink>> action)
66     {
67         using (var scope = new Scope<Types<HeapResizableDirectMemory,
68             ↳ UInt64UnitedMemoryLinks>>())
69         {
70             action(scope.Use<ILinks<TLink>>());
71         }
72     }
73 }

```

Index

`./csharp/Platform.Data.Doublets.Tests/GenericLinksTests.cs`, 454
`./csharp/Platform.Data.Doublets.Tests/LinksConstantsTests.cs`, 456
`./csharp/Platform.Data.Doublets.Tests/ResizableDirectMemoryLinksTests.cs`, 456
`./csharp/Platform.Data.Doublets.Tests/ScopeTests.cs`, 458
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryGenericLinksTests.cs`, 459
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt32LinksTests.cs`, 460
`./csharp/Platform.Data.Doublets.Tests/SplitMemoryUInt64LinksTests.cs`, 462
`./csharp/Platform.Data.Doublets.Tests/TestExtensions.cs`, 463
`./csharp/Platform.Data.Doublets.Tests/UInt64LinksExtensionsTests.cs`, 466
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt32LinksTests.cs`, 467
`./csharp/Platform.Data.Doublets.Tests/UnitedMemoryUInt64LinksTests.cs`, 468
`./csharp/Platform.Data.Doublets/CriterionMatchers/TargetMatcher.cs`, 1
`./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUniquenessAndUsagesResolver.cs`, 1
`./csharp/Platform.Data.Doublets/Decorators/LinksCascadeUsagesResolver.cs`, 2
`./csharp/Platform.Data.Doublets/Decorators/LinksDecoratorBase.cs`, 3
`./csharp/Platform.Data.Doublets/Decorators/LinksDisposableDecoratorBase.cs`, 5
`./csharp/Platform.Data.Doublets/Decorators/LinksInnerReferenceExistenceValidator.cs`, 7
`./csharp/Platform.Data.Doublets/Decorators/LinksItselfConstantToSelfReferenceResolver.cs`, 8
`./csharp/Platform.Data.Doublets/Decorators/LinksNonExistentDependenciesCreator.cs`, 9
`./csharp/Platform.Data.Doublets/Decorators/LinksNullConstantToSelfReferenceResolver.cs`, 10
`./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessResolver.cs`, 11
`./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessValidator.cs`, 12
`./csharp/Platform.Data.Doublets/Decorators/LinksUsagesValidator.cs`, 13
`./csharp/Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.cs`, 14
`./csharp/Platform.Data.Doublets/Decorators/UInt32Links.cs`, 15
`./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`, 27
`./csharp/Platform.Data.Doublets/ILinks.cs`, 27
`./csharp/Platform.Data.Doublets/ILinksExtensions.cs`, 28
`./csharp/Platform.Data.Doublets/ISynchronizedLinks.cs`, 46
`./csharp/Platform.Data.Doublets/Link.cs`, 47
`./csharp/Platform.Data.Doublets/LinkExtensions.cs`, 55
`./csharp/Platform.Data.Doublets/LinksOperatorBase.cs`, 56
`./csharp/Platform.Data.Doublets/Memory/ILinksListMethods.cs`, 56
`./csharp/Platform.Data.Doublets/Memory/ILinksTreeMethods.cs`, 57
`./csharp/Platform.Data.Doublets/Memory/IndexTreeType.cs`, 58
`./csharp/Platform.Data.Doublets/Memory/LinksHeader.cs`, 59
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 61
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.cs`, 68
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 75
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 79
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 83
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 87
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 91
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs`, 97
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesLinkedListMethods.cs`, 102
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 108
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs`, 111
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 115
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksTargetsSizeBalancedTreeMethods.cs`, 119
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs`, 122
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs`, 126
`./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs`, 145
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs`, 148
`./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs`, 149
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 151
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase.cs`, 157
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs`, 163
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMethods.cs`, 167
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs`, 171
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMethods.cs`, 175
`./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs`, 179

./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSizeBalancedTreeMethodsBase.cs, 184
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesLinkedListMethods.cs, 190
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 191
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMethods.cs, 195
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 198
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMethods.cs, 202
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32SplitMemoryLinks.cs, 206
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs, 213
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 214
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSizeBalancedTreeMethodsBase.cs, 220
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 226
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesSizeBalancedTreeMethods.cs, 230
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 234
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMethods.cs, 237
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase.cs, 241
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.cs, 247
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs, 253
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 254
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMethods.cs, 257
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 261
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsSizeBalancedTreeMethods.cs, 265
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs, 268
./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs, 275
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksAvlBalancedTreeMethodsBase.cs, 277
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 286
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSizeBalancedTreeMethodsBase.cs, 293
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesAvlBalancedTreeMethods.cs, 300
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 305
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs, 309
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs, 312
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 318
./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsSizeBalancedTreeMethods.cs, 321
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs, 325
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs, 328
./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs, 341
./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs, 344
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 346
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSizeBalancedTreeMethodsBase.cs, 352
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 357
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesSizeBalancedTreeMethods.cs, 361
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 365
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsSizeBalancedTreeMethods.cs, 369
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnitedMemoryLinks.cs, 372
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs, 378
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksAvlBalancedTreeMethodsBase.cs, 379
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMethodsBase.cs, 387
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSizeBalancedTreeMethodsBase.cs, 392
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.cs, 398
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods.cs, 403
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.cs, 407
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsAvlBalancedTreeMethods.cs, 411
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods.cs, 416
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.cs, 420
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnitedMemoryLinks.cs, 423
./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs, 429
./csharp/Platform.Data.Doublets/PropertyOperators/PropertiesOperator.cs, 430
./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs, 432
./csharp/Platform.Data.Doublets/Stacks/Stack.cs, 434
./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs, 436
./csharp/Platform.Data.Doublets/SynchronizedLinks.cs, 437
./csharp/Platform.Data.Doublets/UInt64LinksExtensions.cs, 439
./csharp/Platform.Data.Doublets/UInt64LinksTransactionsLayer.cs, 443