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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Interfaces;

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

namespace Platform.Data.Doublets.CriterionMatchers
{
    /// <summary>
    /// <para>
    /// Represents the target matcher.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksOperatorBase{TLink}"/>
    /// <seealso cref="ICriterionMatcher{TLink}"/>
    public class TargetMatcher<TLink> : LinksOperatorBase<TLink>, ICriterionMatcher<TLink>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        private readonly TLink _targetToMatch;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="TargetMatcher"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="targetToMatch">
        /// <para>A target to match.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TargetMatcher(ILinks<TLink> links, TLink targetToMatch) : base(links) =>
        ↪  _targetToMatch = targetToMatch;

        /// <summary>
        /// <para>
        /// Determines whether this instance is matched.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool IsMatched(TLink link) => _equalityComparer.Equals(_links.GetTarget(link),
        ↪  _targetToMatch);
    }
}
```

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the links cascade uniqueness and usages resolver.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksUniquenessResolver{TLink}"/>
    public class LinksCascadeUniquenessAndUsagesResolver<TLink> : LinksUniquenessResolver<TLink>
    {
```

```
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksCascadeUniquenessAndUsagesResolver"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksCascadeUniquenessAndUsagesResolver(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Resolves the address change conflict using the specified old link address.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="oldLinkAddress">
        /// <para>The old link address.</para>
        /// <para></para>
        /// </param>
        /// <param name="newLinkAddress">
        /// <para>The new link address.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
        ↪  newLinkAddress)
        {
            // Use Facade (the last decorator) to ensure recursion working correctly
            _facade.MergeUsages(oldLinkAddress, newLinkAddress);
            return base.ResolveAddressChangeConflict(oldLinkAddress, newLinkAddress);
        }
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <remarks>
    /// <para>Must be used in conjunction with NonNullContentsLinkDeletionResolver.</para>
    /// <para>Должен использоваться вместе с NonNullContentsLinkDeletionResolver.</para>
    /// </remarks>
    public class LinksCascadeUsagesResolver<TLink> : LinksDecoratorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksCascadeUsagesResolver"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksCascadeUsagesResolver(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Deletes the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
```

```
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override void Delete(IList<TLink> restrictions)
        {
            var linkIndex = restrictions[_constants.IndexPart];
            // Use Facade (the last decorator) to ensure recursion working correctly
            _facade.DeleteAllUsages(linkIndex);
            _links.Delete(linkIndex);
        }
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the links decorator base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksOperatorBase{TLink}"/>
    /// <seealso cref="ILinks{TLink}"/>
    public abstract class LinksDecoratorBase<TLink> : LinksOperatorBase<TLink>, ILinks<TLink>
    {
        /// <summary>
        /// <para>
        /// The constants.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly LinksConstants<TLink> _constants;

        /// <summary>
        /// <para>
        /// Gets the constants value.
        /// </para>
        /// <para></para>
        /// </summary>
        public LinksConstants<TLink> Constants
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _constants;
        }

        /// <summary>
        /// <para>
        /// The facade.
        /// </para>
        /// <para></para>
        /// </summary>
        protected ILinks<TLink> _facade;

        /// <summary>
        /// <para>
        /// Gets or sets the facade value.
        /// </para>
        /// <para></para>
        /// </summary>
        public ILinks<TLink> Facade
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _facade;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set
            {
                _facade = value;
                if (_links is LinksDecoratorBase<TLink> decorator)
                {
                    decorator.Facade = value;
                }
            }
        }
```

```
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)
            ↪ => _links.Each(handler, restrictions);
122
123         /// <summary>
124         /// <para>
125         /// Creates the restrictions.
126         /// </para>
127         /// <para></para>
128         /// </summary>
129         /// <param name="restrictions">
130         /// <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(IList<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(IList<TLink> restrictions, IList<TLink> substitution) =>
            ↪ _links.Update(restrictions, substitution);
160
161         /// <summary>
162         /// <para>
163         /// Deletes the restrictions.
164         /// </para>
165         /// <para></para>
166         /// </summary>
167         /// <param name="restrictions">
168         /// <para>The restrictions.</para>
169         /// <para></para>
170         /// </param>
171         [MethodImpl(MethodImplOptions.AggressiveInlining)]
172         public virtual void Delete(IList<TLink> restrictions) => _links.Delete(restrictions);
173     }
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>,
       ↪ ILinks<TLink>, System.IDisposable
19     {
20         /// <summary>
21         /// <para>
22         /// Represents the disposable with multiple calls allowed.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <seealso cref="Disposable"/>
27         protected class DisposableWithMultipleCallsAllowed : Disposable
28         {
29             /// <summary>
30             /// <para>
31             /// Initializes a new <see cref="DisposableWithMultipleCallsAllowed"/> instance.
32             /// </para>
33             /// <para></para>
34             /// </summary>
35             /// <param name="disposal">
36             /// <para>A disposal.</para>
37             /// <para></para>
38             /// </param>
39             [MethodImpl(MethodImplOptions.AggressiveInlining)]
40             public DisposableWithMultipleCallsAllowed(Disposal disposal) : base(disposal) { }
41
42             /// <summary>
43             /// <para>
```

```
            /// Gets the allow multiple dispose calls value.
            /// </para>
            /// <para></para>
            /// </summary>
            protected override bool AllowMultipleDisposeCalls
            {
                [MethodImpl(MethodImplOptions.AggressiveInlining)]
                get => true;
            }
        }

        /// <summary>
        /// <para>
        /// The disposable.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly DisposableWithMultipleCallsAllowed Disposable;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksDisposableDecoratorBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected LinksDisposableDecoratorBase(ILinks<TLink> links) : base(links) => Disposable
        ↪  = new DisposableWithMultipleCallsAllowed(Dispose);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        ~LinksDisposableDecoratorBase() => Disposable.Destruct();

        /// <summary>
        /// <para>
        /// Disposes this instance.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Dispose() => Disposable.Dispose();

        /// <summary>
        /// <para>
        /// Disposes the manual.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="manual">
        /// <para>The manual.</para>
        /// <para></para>
        /// </param>
        /// <param name="wasDisposed">
        /// <para>The was disposed.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                _links.DisposeIfPossible();
            }
        }
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
```

```
    // TODO: Make LinksExternalReferenceValidator. A layer that checks each link to exist or to be external (hybrid link's raw number).
    /// <summary>
    /// <para>
    /// Represents the links inner reference existence validator.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDecoratorBase{TLink}"/>
    public class LinksInnerReferenceExistenceValidator<TLink> : LinksDecoratorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksInnerReferenceExistenceValidator"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksInnerReferenceExistenceValidator(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Eaches the handler.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
        {
            var links = _links;
            links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
            return links.Each(handler, restrictions);
        }

        /// <summary>
        /// <para>
        /// Updates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitution">
        /// <para>The substitution.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
        {
            // TODO: Possible values: null, ExistentLink or NonExistentHybrid(ExternalReference)
            var links = _links;
            links.EnsureInnerReferenceExists(restrictions, nameof(restrictions));
            links.EnsureInnerReferenceExists(substitution, nameof(substitution));
            return links.Update(restrictions, substitution);
        }

```

```
        /// <summary>
        /// <para>
        /// Deletes the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override void Delete(IList<TLink> restrictions)
        {
            var link = restrictions[_constants.IndexPart];
            var links = _links;
            links.EnsureLinkExists(link, nameof(link));
            links.Delete(link);
        }
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the links itself constant to self reference resolver.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDecoratorBase{TLink}"/>
    public class LinksItselfConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksItselfConstantToSelfReferenceResolver"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksItselfConstantToSelfReferenceResolver(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Eaches the handler.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
        {
            var constants = _constants;
            var itselfConstant = constants.Itself;
```

```
            if (!_equalityComparer.Equals(constants.Any, itselfConstant) &&
            ↪  restrictions.Contains(itselfConstant))
            {
                // Itself constant is not supported for Each method right now, skipping execution
                return constants.Continue;
            }
            return _links.Each(handler, restrictions);
        }

        /// <summary>
        /// <para>
        /// Updates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitution">
        /// <para>The substitution.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
        ↪  _links.Update(restrictions, _links.ResolveConstantAsSelfReference(_constants.Itself,
        ↪  restrictions, substitution));
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <remarks>
    /// Not practical if newSource and newTarget are too big.
    /// To be able to use practical version we should allow to create link at any specific
    ↪  location inside ResizableDirectMemoryLinks.
    /// This in turn will require to implement not a list of empty links, but a list of ranges
    ↪  to store it more efficiently.
    /// </remarks>
    public class LinksNonExistentDependenciesCreator<TLink> : LinksDecoratorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksNonExistentDependenciesCreator"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksNonExistentDependenciesCreator(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Updates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitution">
        /// <para>The substitution.</para>
        /// <para></para>
        /// </param>
```

```
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
        {
            var constants = _constants;
            var links = _links;
            links.EnsureCreated(substitution[constants.SourcePart],
            ↪  substitution[constants.TargetPart]);
            return links.Update(restrictions, substitution);
        }
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the links null constant to self reference resolver.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDecoratorBase{TLink}"/>
    public class LinksNullConstantToSelfReferenceResolver<TLink> : LinksDecoratorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksNullConstantToSelfReferenceResolver"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksNullConstantToSelfReferenceResolver(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Creates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Create(IList<TLink> restrictions) => _links.CreatePoint();

        /// <summary>
        /// <para>
        /// Updates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitution">
        /// <para>The substitution.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

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

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

```
1   using System.Collections.Generic;
2   using System.Runtime.CompilerServices;
3
4   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6   namespace Platform.Data.Doublets.Decorators
7   {
8       /// <summary>
9       /// <para>
10      /// Represents the links uniqueness resolver.
11      /// </para>
12      /// <para></para>
13      /// </summary>
14      /// <seealso cref="LinksDecoratorBase{TLink}"/>
15      public class LinksUniquenessResolver<TLink> : LinksDecoratorBase<TLink>
16      {
17          private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
18
19          /// <summary>
20          /// <para>
21          /// Initializes a new <see cref="LinksUniquenessResolver"/> instance.
22          /// </para>
23          /// <para></para>
24          /// </summary>
25          /// <param name="links">
26          /// <para>A links.</para>
27          /// <para></para>
28          /// </param>
29          [MethodImpl(MethodImplOptions.AggressiveInlining)]
30          public LinksUniquenessResolver(ILinks<TLink> links) : base(links) { }
31
32          /// <summary>
33          /// <para>
34          /// Updates the restrictions.
35          /// </para>
36          /// <para></para>
37          /// </summary>
38          /// <param name="restrictions">
39          /// <para>The restrictions.</para>
40          /// <para></para>
41          /// </param>
42          /// <param name="substitution">
43          /// <para>The substitution.</para>
44          /// <para></para>
45          /// </param>
46          /// <returns>
47          /// <para>The link</para>
48          /// <para></para>
49          /// </returns>
50          [MethodImpl(MethodImplOptions.AggressiveInlining)]
51          public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
52          {
53              var constants = _constants;
54              var links = _links;
55              var newLinkAddress = links.SearchOrDefault(substitution[constants.SourcePart],
                ↪ substitution[constants.TargetPart]);
56              if (_equalityComparer.Equals(newLinkAddress, default))
57              {
58                  return links.Update(restrictions, substitution);
59              }
60              return ResolveAddressChangeConflict(restrictions[constants.IndexPart],
                ↪ newLinkAddress);
61          }
62
63          /// <summary>
64          /// <para>
```

```
        /// Resolves the address change conflict using the specified old link address.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="oldLinkAddress">
        /// <para>The old link address.</para>
        /// <para></para>
        /// </param>
        /// <param name="newLinkAddress">
        /// <para>The new link address.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The new link address.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink ResolveAddressChangeConflict(TLink oldLinkAddress, TLink
        ↪  newLinkAddress)
        {
            if (!_equalityComparer.Equals(oldLinkAddress, newLinkAddress) &&
            ↪  _links.Exists(oldLinkAddress))
            {
                _facade.Delete(oldLinkAddress);
            }
            return newLinkAddress;
        }
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the links uniqueness validator.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDecoratorBase{TLink}"/>
    public class LinksUniquenessValidator<TLink> : LinksDecoratorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksUniquenessValidator"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksUniquenessValidator(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Updates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitution">
        /// <para>The substitution.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
```

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
        {
            var links = _links;
            var constants = _constants;
            links.EnsureDoesNotExists(substitution[constants.SourcePart],
            ↪  substitution[constants.TargetPart]);
            return links.Update(restrictions, substitution);
        }
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the links usages validator.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDecoratorBase{TLink}"/>
    public class LinksUsagesValidator<TLink> : LinksDecoratorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksUsagesValidator"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksUsagesValidator(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Updates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitution">
        /// <para>The substitution.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
        {
            var links = _links;
            links.EnsureNoUsages(restrictions[_constants.IndexPart]);
            return links.Update(restrictions, substitution);
        }

        /// <summary>
        /// <para>
        /// Deletes the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
        public override void Delete(IList<TLink> restrictions)
        {
            var link = restrictions[_constants.IndexPart];
            var links = _links;
            links.EnsureNoUsages(link);
            links.Delete(link);
        }
    }
}
```

1.13 ./csharp/Platform.Data.Doublets/Decorators/NonNullContentsLinkDeletionResolver.cs

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the non null contents link deletion resolver.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDecoratorBase{TLink}"/>
    public class NonNullContentsLinkDeletionResolver<TLink> : LinksDecoratorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="NonNullContentsLinkDeletionResolver"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public NonNullContentsLinkDeletionResolver(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Deletes the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override void Delete(IList<TLink> restrictions)
        {
            var linkIndex = restrictions[_constants.IndexPart];
            var links = _links;
            links.EnforceResetValues(linkIndex);
            links.Delete(linkIndex);
        }
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using TLink = System.UInt32;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the int 32 links.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDisposableDecoratorBase{TLink}"/>
    public class UInt32Links : LinksDisposableDecoratorBase<TLink>
    {
```

```
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32Links"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt32Links(ILinks<TLink> links) : base(links) { }

        /// <summary>
        /// <para>
        /// Creates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Create(IList<TLink> restrictions) => _links.CreatePoint();

        /// <summary>
        /// <para>
        /// Updates the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitution">
        /// <para>The substitution.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
        {
            var constants = _constants;
            var indexPartConstant = constants.IndexPart;
            var sourcePartConstant = constants.SourcePart;
            var targetPartConstant = constants.TargetPart;
            var nullConstant = constants.Null;
            var itselfConstant = constants.Itself;
            var existedLink = nullConstant;
            var updatedLink = restrictions[indexPartConstant];
            var newSource = substitution[sourcePartConstant];
            var newTarget = substitution[targetPartConstant];
            var links = _links;
            if (newSource != itselfConstant && newTarget != itselfConstant)
            {
                existedLink = links.SearchOrDefault(newSource, newTarget);
            }
            if (existedLink == nullConstant)
            {
                var before = links.GetLink(updatedLink);
                if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
                ↪  newTarget)
                {
                    links.Update(updatedLink, newSource == itselfConstant ? updatedLink :
                    ↪  newSource,
                                              newTarget == itselfConstant ? updatedLink :
                                              ↪  newTarget);
                }
                return updatedLink;
```

```
            }
            else
            {
                return _facade.MergeAndDelete(updatedLink, existedLink);
            }
        }

        /// <summary>
        /// <para>
        /// Deletes the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override void Delete(IList<TLink> restrictions)
        {
            var linkIndex = restrictions[_constants.IndexPart];
            var links = _links;
            links.EnforceResetValues(linkIndex);
            _facade.DeleteAllUsages(linkIndex);
            links.Delete(linkIndex);
        }
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>Represents a combined decorator that implements the basic logic for interacting with the links storage for links with addresses represented as <see cref="System.UInt64" />.</para>
    /// <para>Представляет комбинированный декоратор, реализующий основную логику по взаимодействии с хранилищем связей, для связей с адресами представленными в виде <see cref="System.UInt64"/>.</para>
    /// </summary>
    /// <remarks>
    /// Возможные оптимизации:
    /// Объединение в одном поле Source и Target с уменьшением до 32 бит.
    ///     + меньше объём БД
    ///     - меньше производительность
    ///     - больше ограничение на количество связей в БД)
    /// Ленивое хранение размеров поддеревьев (расчитываемое по мере использования БД)
    ///     + меньше объём БД
    ///     - больше сложность
    ///
    /// 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
    /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами (битовыми строками) - вариант матрицы (выстраеваемой лениво).
    ///
    /// Решить отключать ли проверки при компиляции под Release. Т.е. исключения будут выбрасываться только при #if DEBUG
    /// </remarks>
    public class UInt64Links : LinksDisposableDecoratorBase<ulong>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64Links"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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              if (existedLink == nullConstant)
96              {
97                  var before = links.GetLink(updatedLink);
98                  if (before[sourcePartConstant] != newSource || before[targetPartConstant] !=
                        ↪ newTarget)
99                  {
100                     links.Update(updatedLink, newSource == itselfConstant ? updatedLink :
                            ↪ newSource,
101                                              newTarget == itselfConstant ? updatedLink :
                                                    ↪ newTarget);
102                 }
103                 return updatedLink;
104             }
105             else
106             {
107                 return _facade.MergeAndDelete(updatedLink, existedLink);
108             }
109         }
110
111         /// <summary>
112         /// <para>
113         /// Deletes the restrictions.
114         /// </para>
115         /// <para></para>
```

```
        /// </summary>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override void Delete(IList<ulong> restrictions)
        {
            var linkIndex = restrictions[_constants.IndexPart];
            var links = _links;
            links.EnforceResetValues(linkIndex);
            _facade.DeleteAllUsages(linkIndex);
            links.Delete(linkIndex);
        }
    }
}
```

1.16 ./csharp/Platform.Data.Doublets/Decorators/UniLinks.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Platform.Collections;
using Platform.Collections.Lists;
using Platform.Data.Universal;

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

namespace Platform.Data.Doublets.Decorators
{
    /// <remarks>
    /// What does empty pattern (for condition or substitution) mean? Nothing or Everything?
    /// Now we go with nothing. And nothing is something one, but empty, and cannot be changed
    ↪  by itself. But can cause creation (update from nothing) or deletion (update to nothing).
    ///
    /// TODO: Decide to change to IDoubletLinks or not to change. (Better to create
    ↪  DefaultUniLinksBase, that contains logic itself and can be implemented using both
    ↪  IDoubletLinks and ILinks.)
    /// </remarks>
    internal class UniLinks<TLink> : LinksDecoratorBase<TLink>, IUniLinks<TLink>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UniLinks"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        public UniLinks(ILinks<TLink> links) : base(links) { }

        private struct Transition
        {
            /// <summary>
            /// <para>
            /// The before.
            /// </para>
            /// <para></para>
            /// </summary>
            public IList<TLink> Before;
            /// <summary>
            /// <para>
            /// The after.
            /// </para>
            /// <para></para>
            /// </summary>
            public IList<TLink> After;

            /// <summary>
            /// <para>
            /// Initializes a new <see cref="Transition"/> instance.
            /// </para>
            /// <para></para>
            /// </summary>
            /// <param name="before">
```

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

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

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

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

```
348                 {
349                     var linkToDelete = patternOrCondition[0];
350                     var before = _links.GetLink(linkToDelete);
351                     if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
                        ↪  constants.Break))
352                     {
353                         return constants.Break;
354                     }
355                     var after = Array.Empty<TLink>();
356                     _links.Update(linkToDelete, constants.Null, constants.Null);
357                     _links.Delete(linkToDelete);
358                     if (matchHandler != null)
359                     {
360                         return substitutionHandler(before, after);
361                     }
362                     return constants.Continue;
363                 }
364                 else
365                 {
366                     throw new NotSupportedException();
367                 }
368             }
369             else // Replace / Update
370             {
371                 if (patternOrCondition.Count == 1) //-V3125
372                 {
373                     var linkToUpdate = patternOrCondition[0];
374                     var before = _links.GetLink(linkToUpdate);
375                     if (matchHandler != null && _equalityComparer.Equals(matchHandler(before),
                        ↪  constants.Break))
376                     {
377                         return constants.Break;
378                     }
379                     var after = (IList<TLink>)substitution.ToArray(); //-V3125
380                     if (_equalityComparer.Equals(after[0], default))
381                     {
382                         after[0] = linkToUpdate;
383                     }
384                     if (substitution.Count == 1)
385                     {
386                         if (!_equalityComparer.Equals(substitution[0], linkToUpdate))
387                         {
388                             after = _links.GetLink(substitution[0]);
389                             _links.Update(linkToUpdate, constants.Null, constants.Null);
390                             _links.Delete(linkToUpdate);
391                         }
392                     }
393                     else if (substitution.Count == 3)
394                     {
395                         //Links.Update(after);
396                     }
397                     else
398                     {
399                         throw new NotSupportedException();
400                     }
401                     if (matchHandler != null)
402                     {
403                         return substitutionHandler(before, after);
404                     }
405                     return constants.Continue;
406                 }
407                 else
408                 {
409                     throw new NotSupportedException();
410                 }
411             }
412         }
413 
414         /// <remarks>
415         /// IList[IList[IList[T]]]
416         /// |      |      |      |||
417         /// |      |      ------ ||
418         /// |      |       link  ||
419         /// |      -------------- |
420         /// |         change      |
421         ///  ---------------------
422         ///         changes
423         /// </remarks>
```

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

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

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

```
        {
            Source = source;
            Target = target;
        }

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <returns>
        /// <para>.</para>
        /// <para>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override string ToString() => $"{Source}->{Target}";

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <typeparam>
        /// <para>.</para>
        /// <para>.</para>
        /// </typeparam>
        /// <param name="other">
        /// <para>.</para>
        /// <para>.</para>
        /// </param>
        /// <returns>
        /// <para>.</para>
        /// <para>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Equals(Doublet<T> other) => _equalityComparer.Equals(Source, other.Source)
        ↪  && _equalityComparer.Equals(Target, other.Target);

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <typeparam>
        /// <para>.</para>
        /// <para>.</para>
        /// </typeparam>
        /// <param name="obj">
        /// <para>.</para>
        /// <para>.</para>
        /// </param>
        /// <returns>
        /// <para>.</para>
        /// <para>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override bool Equals(object obj) => obj is Doublet<T> doublet ?
        ↪  base.Equals(doublet) : false;

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <returns>
        /// <para>.</para>
        /// <para>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override int GetHashCode() => (Source, Target).GetHashCode();

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <param name="left">
        /// <para>.</para>
        /// <para>.</para>
        /// </param>
        /// <param name="right">
        /// <para>.</para>
        /// <para>.</para>
```

```
        /// </param>
        /// <returns>
        /// <para>.</para>
        /// <para>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool operator ==(Doublet<T> left, Doublet<T> right) => left.Equals(right);

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <param name="left">
        /// <para>.</para>
        /// <para>.</para>
        /// </param>
        /// <param name="right">
        /// <para>.</para>
        /// <para>.</para>
        /// </param>
        /// <returns>
        /// <para>.</para>
        /// <para>.</para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool operator !=(Doublet<T> left, Doublet<T> right) => !(left == right);
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets
{
    /// <remarks>
    /// TODO: Может стоит попробовать ref во всех методах (IRefEqualityComparer)
    /// 2x faster with comparer
    /// </remarks>
    public class DoubletComparer<T> : IEqualityComparer<Doublet<T>>
    {
        /// <summary>
        /// <para>
        /// The .
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly DoubletComparer<T> Default = new DoubletComparer<T>();

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="x">
        /// <para>The .</para>
        /// <para></para>
        /// </param>
        /// <param name="y">
        /// <para>The .</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Equals(Doublet<T> x, Doublet<T> y) => x.Equals(y);

        /// <summary>
        /// <para>
        /// Gets the hash code using the specified obj.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
49          /// <param name="obj">
50          /// <para>The obj.</para>
51          /// <para></para>
52          /// </param>
53          /// <returns>
54          /// <para>The int</para>
55          /// <para></para>
56          /// </returns>
57          [MethodImpl(MethodImplOptions.AggressiveInlining)]
58          public int GetHashCode(Doublet<T> obj) => obj.GetHashCode();
59      }
60  }
```

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

```
1   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3   using System.Collections.Generic;
4
5   namespace Platform.Data.Doublets
6   {
7       /// <summary>
8       /// <para>
9       /// Defines the links.
10      /// </para>
11      /// <para></para>
12      /// </summary>
13      /// <seealso cref="ILinks{TLink, LinksConstants{TLink}}"/>
14      public interface ILinks<TLink> : ILinks<TLink, LinksConstants<TLink>>
15      {
16      }
17  }
```

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

```
1   using System;
2   using System.Collections;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Runtime.CompilerServices;
6   using Platform.Ranges;
7   using Platform.Collections.Arrays;
8   using Platform.Random;
9   using Platform.Setters;
10  using Platform.Converters;
11  using Platform.Numbers;
12  using Platform.Data.Exceptions;
13  using Platform.Data.Doublets.Decorators;
14
15  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
16
17  namespace Platform.Data.Doublets
18  {
19      /// <summary>
20      /// <para>
21      /// Represents the links extensions.
22      /// </para>
23      /// <para></para>
24      /// </summary>
25      public static class ILinksExtensions
26      {
27          /// <summary>
28          /// <para>
29          /// Runs the random creations using the specified links.
30          /// </para>
31          /// <para></para>
32          /// </summary>
33          /// <typeparam name="TLink">
34          /// <para>The link.</para>
35          /// <para></para>
36          /// </typeparam>
37          /// <param name="links">
38          /// <para>The links.</para>
39          /// <para></para>
40          /// </param>
41          /// <param name="amountOfCreations">
42          /// <para>The amount of creations.</para>
43          /// <para></para>
44          /// </param>
45          [MethodImpl(MethodImplOptions.AggressiveInlining)]
46          public static void RunRandomCreations<TLink>(this ILinks<TLink> links, ulong
            ↪  amountOfCreations)
```

```
47          {
48              var random = RandomHelpers.Default;
49              var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
50              var uInt64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
51              for (var i = 0UL; i < amountOfCreations; i++)
52              {
53                  var linksAddressRange = new Range<ulong>(0,
                    ↪  addressToUInt64Converter.Convert(links.Count()));
54                  var source =
                    ↪  uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
55                  var target =
                    ↪  uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
56                  links.GetOrCreate(source, target);
57              }
58          }
59
60          /// <summary>
61          /// <para>
62          /// Runs the random searches using the specified links.
63          /// </para>
64          /// <para></para>
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;
```

```
            var linksCount = addressToUInt64Converter.Convert(links.Count());
            var min = amountOfDeletions > linksCount ? OUL : linksCount - amountOfDeletions;
            for (var i = OUL; i < amountOfDeletions; i++)
            {
                linksCount = addressToUInt64Converter.Convert(links.Count());
                if (linksCount <= min)
                {
                    break;
                }
                var linksAddressRange = new Range<ulong>(min, linksCount);
                var link =
                ↪  uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
                links.Delete(link);
            }
        }

        /// <summary>
        /// <para>
        /// Deletes the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="linkToDelete">
        /// <para>The link to delete.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void Delete<TLink>(this ILinks<TLink> links, TLink linkToDelete) =>
        ↪  links.Delete(new LinkAddress<TLink>(linkToDelete));

        /// <remarks>
        /// TODO: Возможно есть очень простой способ это сделать.
        /// (Например просто удалить файл, или изменить его размер таким образом,
        /// чтобы удалился весь контент)
        /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
        /// </remarks>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void DeleteAll<TLink>(this ILinks<TLink> links)
        {
            var equalityComparer = EqualityComparer<TLink>.Default;
            var comparer = Comparer<TLink>.Default;
            for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
            ↪  Arithmetic.Decrement(i))
            {
                links.Delete(i);
                if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
                {
                    i = links.Count();
                }
            }
        }

        /// <summary>
        /// <para>
        /// Firsts the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="InvalidOperationException">
        /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
        /// <para></para>
        /// </exception>
```

```
192         /// <exception cref="InvalidOperationException">
193         /// <para>В хранилище нет связей.</para>
194         /// <para></para>
195         /// </exception>
196         /// <returns>
197         /// <para>The first link.</para>
198         /// <para></para>
199         /// </returns>
200         [MethodImpl(MethodImplOptions.AggressiveInlining)]
201         public static TLink First<TLink>(this ILinks<TLink> links)
202         {
203             TLink firstLink = default;
204             var equalityComparer = EqualityComparer<TLink>.Default;
205             if (equalityComparer.Equals(links.Count(), default))
206             {
207                 throw new InvalidOperationException("В хранилище нет связей.");
208             }
209             links.Each(links.Constants.Any, links.Constants.Any, link =>
210             {
211                 firstLink = link[links.Constants.IndexPart];
212                 return links.Constants.Break;
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 GetByKeys<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>
```

```
        /// </param>
        /// <param name="root">
        /// <para>The root.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        /// <param name="index">
        /// <para>The index.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="ArgumentOutOfRangeException">
        /// <para>Sequences with sizes other than powers of two are not supported.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The current link.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TLink GetSquareMatrixSequenceElementByIndex<TLink>(this ILinks<TLink>
        ↪  links, TLink root, ulong size, ulong index)
        {
            var constants = links.Constants;
            var source = constants.SourcePart;
            var target = constants.TargetPart;
            if (!Platform.Numbers.Math.IsPowerOfTwo(size))
            {
                throw new ArgumentOutOfRangeException(nameof(size), "Sequences with sizes other
                ↪  than powers of two are not supported.");
            }
            var path = new BitArray(BitConverter.GetBytes(index));
            var length = Bit.GetLowestPosition(size);
            links.EnsureLinkExists(root, "root");
            var currentLink = root;
            for (var i = length - 1; i >= 0; i--)
            {
                currentLink = links.GetLink(currentLink)[path[i] ? target : source];
            }
            return currentLink;
        }

        #endregion

        /// <summary>
        /// Возвращает индекс указанной связи.
        /// </summary>
        /// <param name="links">Хранилище связей.</param>
        /// <param name="link">Связь представленная списком, состоящим из её адреса и
        ↪  содержимого.</param>
        /// <returns>Индекс начальной связи для указанной связи.</returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TLink GetIndex<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
        ↪  link[links.Constants.IndexPart];

        /// <summary>
        /// Возвращает индекс начальной (Source) связи для указанной связи.
        /// </summary>
        /// <param name="links">Хранилище связей.</param>
        /// <param name="link">Индекс связи.</param>
        /// <returns>Индекс начальной связи для указанной связи.</returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TLink GetSource<TLink>(this ILinks<TLink> links, TLink link) =>
        ↪  links.GetLink(link)[links.Constants.SourcePart];

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

```

```
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
439         /// <summary>
440         /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
            ↪ (handler) для каждой подходящей связи.
441         /// </summary>
442         /// <param name="links">Хранилище связей.</param>
443         /// <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>
444         /// <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>
445         /// <param name="handler">Обработчик каждой подходящей связи.</param>
446         /// <returns>True, в случае если проход по связям не был прерван и False в обратном
            ↪ случае.</returns>
447         [MethodImpl(MethodImplOptions.AggressiveInlining)]
448         public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
            ↪ Func<TLink, bool> handler)
449         {
450             var constants = links.Constants;
451             return links.Each(link => handler(link[constants.IndexPart]) ? constants.Continue :
                ↪ constants.Break, constants.Any, source, target);
452         }
453
454         /// <summary>
455         /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
            ↪ (handler) для каждой подходящей связи.
456         /// </summary>
457         /// <param name="links">Хранилище связей.</param>
458         /// <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>
459         /// <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>
460         /// <param name="handler">Обработчик каждой подходящей связи.</param>
461         /// <returns>True, в случае если проход по связям не был прерван и False в обратном
            ↪ случае.</returns>
462         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
463         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);
464
465         /// <summary>
466         /// <para>
467         /// Alls the links.
468         /// </para>
469         /// <para></para>
470         /// </summary>
471         /// <typeparam name="TLink">
472         /// <para>The link.</para>
473         /// <para></para>
474         /// </typeparam>
475         /// <param name="links">
476         /// <para>The links.</para>
477         /// <para></para>
478         /// </param>
479         /// <param name="restrictions">
480         /// <para>The restrictions.</para>
481         /// <para></para>
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 
553         #region Ensure
554         // TODO: May be move to EnsureExtensions or make it both there and here
555 
556         /// <summary>
557         /// <para>
558         /// Ensures the link exists using the specified links.
559         /// </para>
560         /// <para></para>
561         /// </summary>
562         /// <typeparam name="TLink">
563         /// <para>The link.</para>
564         /// <para></para>
565         /// </typeparam>
566         /// <param name="links">
567         /// <para>The links.</para>
568         /// <para></para>
569         /// </param>
570         /// <param name="restrictions">
571         /// <para>The restrictions.</para>
572         /// <para></para>
573         /// </param>
574         /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
575         /// <para>sequence[{i}]</para>
576         /// <para></para>
577         /// </exception>
578         [MethodImpl(MethodImplOptions.AggressiveInlining)]
579         public static void EnsureLinkExists<TLink>(this ILinks<TLink> links, IList<TLink>
    ↪   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],
    ↪   $"sequence[{i}]");
586                 }
587             }
588         }
589 
590         /// <summary>
591         /// <para>
592         /// Ensures the inner reference exists using the specified links.
593         /// </para>
594         /// <para></para>
595         /// </summary>
596         /// <typeparam name="TLink">
597         /// <para>The link.</para>
598         /// <para></para>
599         /// </typeparam>
600         /// <param name="links">
601         /// <para>The links.</para>
602         /// <para></para>
603         /// </param>
604         /// <param name="reference">
605         /// <para>The reference.</para>
```

```
        /// <para></para>
        /// </param>
        /// <param name="argumentName">
        /// <para>The argument name.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
        /// <para></para>
        /// <para></para>
        /// </exception>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links, TLink
        ↪  reference, string argumentName)
        {
            if (links.Constants.IsInternalReference(reference) && !links.Exists(reference))
            {
                throw new ArgumentLinkDoesNotExistsException<TLink>(reference, argumentName);
            }
        }

        /// <summary>
        /// <para>
        /// Ensures the inner reference exists using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <param name="argumentName">
        /// <para>The argument name.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnsureInnerReferenceExists<TLink>(this ILinks<TLink> links,
        ↪  IList<TLink> restrictions, string argumentName)
        {
            for (int i = 0; i < restrictions.Count; i++)
            {
                links.EnsureInnerReferenceExists(restrictions[i], argumentName);
            }
        }

        /// <summary>
        /// <para>
        /// Ensures the link is any or exists using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="ArgumentLinkDoesNotExistsException{TLink}">
        /// <para>sequence[{i}]</para>
        /// <para></para>
        /// </exception>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnsureLinkIsAnyOrExists<TLink>(this ILinks<TLink> links, IList<TLink>
        ↪  restrictions)
```

```
680         {
681             var equalityComparer = EqualityComparer<TLink>.Default;
682             var any = links.Constants.Any;
683             for (var i = 0; i < restrictions.Count; i++)
684             {
685                 if (!equalityComparer.Equals(restrictions[i], any) &&
                        !links.Exists(restrictions[i]))
686                 {
687                     throw new ArgumentLinkDoesNotExistsException<TLink>(restrictions[i],
                            $"sequence[{i}]");
688                 }
689             }
690         }
691 
692         /// <summary>
693         /// <para>
694         /// Ensures the link is any or exists using the specified links.
695         /// </para>
696         /// <para></para>
697         /// </summary>
698         /// <typeparam name="TLink">
699         /// <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) =>
            ↪ Comparer<TLink>.Default.Compare(links.CountUsages(link), default) > 0;
845 
846         /// <param name="links">Хранилище связей.</param>
847         [MethodImpl(MethodImplOptions.AggressiveInlining)]
848         public static bool Equals<TLink>(this ILinks<TLink> links, TLink link, TLink source,
            ↪ TLink target)
849         {
850             var constants = links.Constants;
851             var values = links.GetLink(link);
852             var equalityComparer = EqualityComparer<TLink>.Default;
853             return equalityComparer.Equals(values[constants.SourcePart], source) &&
                ↪ equalityComparer.Equals(values[constants.TargetPart], target);
854         }
855 
856         /// <summary>
857         /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
858         /// </summary>
859         /// <param name="links">Хранилище связей.</param>
860         /// <param name="source">Индекс связи, которая является началом для искомой
            ↪ связи.</param>
861         /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
862         /// <returns>Индекс искомой связи с указанными Source (началом) и Target
            ↪ (концом).</returns>
863         [MethodImpl(MethodImplOptions.AggressiveInlining)]
864         public static TLink SearchOrDefault<TLink>(this ILinks<TLink> links, TLink source, TLink
            ↪ target)
865         {
866             var contants = links.Constants;
867             var setter = new Setter<TLink, TLink>(contants.Continue, contants.Break, default);
868             links.Each(setter.SetFirstAndReturnFalse, contants.Any, source, target);
869             return setter.Result;
870         }
871 
872         /// <param name="links">Хранилище связей.</param>
873         [MethodImpl(MethodImplOptions.AggressiveInlining)]
874         public static TLink Create<TLink>(this ILinks<TLink> links) => links.Create(null);
875 
876         /// <param name="links">Хранилище связей.</param>
877         [MethodImpl(MethodImplOptions.AggressiveInlining)]
878         public static TLink CreatePoint<TLink>(this ILinks<TLink> links)
879         {
880             var link = links.Create();
881             return links.Update(link, link, link);
882         }
883 
884         /// <param name="links">Хранилище связей.</param>
885         [MethodImpl(MethodImplOptions.AggressiveInlining)]
886         public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
            ↪ target) => links.Update(links.Create(), source, target);
887 
888         /// <summary>
889         /// Обновляет связь с указанными началом (Source) и концом (Target)
890         /// на связь с указанными началом (NewSource) и концом (NewTarget).
891         /// </summary>
892         /// <param name="links">Хранилище связей.</param>
```

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

```
961             if (equalityComparer.Equals(substitutionIndex, default))
962             {
963                 substitutionIndex = restrictionsIndex;
964             }
965             var source = substitution[constants.SourcePart];
966             var target = substitution[constants.TargetPart];
967             source = equalityComparer.Equals(source, constant) ? substitutionIndex : source;
968             target = equalityComparer.Equals(target, constant) ? substitutionIndex : target;
969             return new Link<TLink>(substitutionIndex, source, target);
970         }
971 
972         /// <summary>
973         /// Создаёт связь (если она не существовала), либо возвращает индекс существующей связи
        ↪  с указанными Source (началом) и Target (концом).
974         /// </summary>
975         /// <param name="links">Хранилище связей.</param>
976         /// <param name="source">Индекс связи, которая является началом на создаваемой
        ↪  связи.</param>
977         /// <param name="target">Индекс связи, которая является концом для создаваемой
        ↪  связи.</param>
978         /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
979         [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;
```

```
            }
            return default;
        }

        /// <summary>Удаляет несколько связей.</summary>
        /// <param name="links">Хранилище связей.</param>
        /// <param name="deletedLinks">Список адресов связей к удалению.</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void DeleteMany<TLink>(this ILinks<TLink> links, IList<TLink> deletedLinks)
        {
            for (int i = 0; i < deletedLinks.Count; i++)
            {
                links.Delete(deletedLinks[i]);
            }
        }

        /// <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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void DeleteAllUsages<TLink>(this ILinks<TLink> links, TLink linkIndex)
        {
            var anyConstant = links.Constants.Any;
            var usagesAsSourceQuery = new Link<TLink>(anyConstant, linkIndex, anyConstant);
            links.DeleteByQuery(usagesAsSourceQuery);
            var usagesAsTargetQuery = new Link<TLink>(anyConstant, linkIndex, anyConstant);
            links.DeleteByQuery(usagesAsTargetQuery);
        }

        /// <summary>
        /// <para>
        /// Deletes the by query using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="query">
        /// <para>The query.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void DeleteByQuery<TLink>(this ILinks<TLink> links, Link<TLink> query)
        {
            var count = CheckedConverter<TLink, long>.Default.Convert(links.Count(query));
            if (count > 0)
            {
                var queryResult = new TLink[count];
                var queryResultFiller = new ArrayFiller<TLink, TLink>(queryResult, links.Constants.Continue);
                links.Each(queryResultFiller.AddFirstAndReturnConstant, query);
                for (var i = count - 1; i >= 0; i--)
                {
                    links.Delete(queryResult[i]);
                }
            }
        }

        // TODO: Move to Platform.Data
        /// <summary>
        /// <para>
        /// Determines whether are values reset.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
```

```
        /// </param>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool AreValuesReset<TLink>(this ILinks<TLink> links, TLink linkIndex)
        {
            var nullConstant = links.Constants.Null;
            var equalityComparer = EqualityComparer<TLink>.Default;
            var link = links.GetLink(linkIndex);
            for (int i = 1; i < link.Count; i++)
            {
                if (!equalityComparer.Equals(link[i], nullConstant))
                {
                    return false;
                }
            }
            return true;
        }

        // TODO: Create a universal version of this method in Platform.Data (with using of for
        ↪  loop)
        /// <summary>
        /// <para>
        /// Resets the values using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void ResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
        {
            var nullConstant = links.Constants.Null;
            var updateRequest = new Link<TLink>(linkIndex, nullConstant, nullConstant);
            links.Update(updateRequest);
        }

        // TODO: Create a universal version of this method in Platform.Data (with using of for
        ↪  loop)
        /// <summary>
        /// <para>
        /// Enforces the reset values using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnforceResetValues<TLink>(this ILinks<TLink> links, TLink linkIndex)
        {
            if (!links.AreValuesReset(linkIndex))
            {
```

```
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 =
                     ↪  addressToInt64Converter.Convert(links.Count(usagesAsTargetQuery));
1198                 var isStandalonePoint = Point<TLink>.IsFullPoint(links.GetLink(oldLinkIndex)) &&
                     ↪  usagesAsSourceCount == 1 && usagesAsTargetCount == 1;
1199                 if (!isStandalonePoint)
1200                 {
1201                     var totalUsages = usagesAsSourceCount + usagesAsTargetCount;
1202                     if (totalUsages > 0)
1203                     {
1204                         var usages = ArrayPool.Allocate<TLink>(totalUsages);
1205                         var usagesFiller = new ArrayFiller<TLink, TLink>(usages,
                             ↪  links.Constants.Continue);
1206                         var i = 0L;
1207                         if (usagesAsSourceCount > 0)
1208                         {
1209                             links.Each(usagesFiller.AddFirstAndReturnConstant,
                                 ↪  usagesAsSourceQuery);
1210                             for (; i < usagesAsSourceCount; i++)
1211                             {
1212                                 var usage = usages[i];
1213                                 if (!equalityComparer.Equals(usage, oldLinkIndex))
1214                                 {
1215                                     links.Update(usage, newLinkIndex, links.GetTarget(usage));
1216                                 }
1217                             }
1218                         }
1219                         if (usagesAsTargetCount > 0)
1220                         {
1221                             links.Each(usagesFiller.AddFirstAndReturnConstant,
                                 ↪  usagesAsTargetQuery);
1222                             for (; i < usages.Length; i++)
1223                             {
1224                                 var usage = usages[i];
1225                                 if (!equalityComparer.Equals(usage, oldLinkIndex))
1226                                 {
1227                                     links.Update(usage, links.GetSource(usage), newLinkIndex);
1228                                 }
1229                             }
1230                         }
1231                         ArrayPool.Free(usages);
1232                     }
1233                 }
1234             }
1235             return newLinkIndex;
1236         }
1237 
1238         /// <summary>
1239         /// Replace one link with another (replaced link is deleted, children are updated or
             ↪  deleted).
1240         /// </summary>
1241         [MethodImpl(MethodImplOptions.AggressiveInlining)]
1242         public static TLink MergeAndDelete<TLink>(this ILinks<TLink> links, TLink oldLinkIndex,
             ↪  TLink newLinkIndex)
1243         {
1244             var equalityComparer = EqualityComparer<TLink>.Default;
```

```
            if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
            {
                links.MergeUsages(oldLinkIndex, newLinkIndex);
                links.Delete(oldLinkIndex);
            }
            return newLinkIndex;
        }

        /// <summary>
        /// <para>
        /// Decorates the with automatic uniqueness and usages resolution using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The links.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static ILinks<TLink>
        ↪  DecorateWithAutomaticUniquenessAndUsagesResolution<TLink>(this ILinks<TLink> links)
        {
            links = new LinksCascadeUsagesResolver<TLink>(links);
            links = new NonNullContentsLinkDeletionResolver<TLink>(links);
            links = new LinksCascadeUniquenessAndUsagesResolver<TLink>(links);
            return links;
        }

        /// <summary>
        /// <para>
        /// Formats the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string Format<TLink>(this ILinks<TLink> links, IList<TLink> link)
        {
            var constants = links.Constants;
            return $"({link[constants.IndexPart]}: {link[constants.SourcePart]}
            ↪  {link[constants.TargetPart]})";
        }

        /// <summary>
        /// <para>
        /// Formats the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
```

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

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

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

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

```
1   using Platform.Collections.Lists;
2   using Platform.Exceptions;
3   using Platform.Ranges;
4   using Platform.Singletons;
5   using System;
6   using System.Collections;
7   using System.Collections.Generic;
8   using System.Runtime.CompilerServices;
9
10  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12  namespace Platform.Data.Doublets
13  {
14      /// <summary>
15      /// Структура описывающая уникальную связь.
16      /// </summary>
17      public struct Link<TLink> : IEquatable<Link<TLink>>, IReadOnlyList<TLink>, IList<TLink>
18      {
19          /// <summary>
20          /// <para>
21          /// The link.
22          /// </para>
23          /// <para></para>
24          /// </summary>
25          public static readonly Link<TLink> Null = new Link<TLink>();
26
27          private static readonly LinksConstants<TLink> _constants =
            ↪ Default<LinksConstants<TLink>>.Instance;
28          private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪ EqualityComparer<TLink>.Default;
29
30          private const int Length = 3;
31
32          /// <summary>
33          /// <para>
34          /// The index.
35          /// </para>
36          /// <para></para>
37          /// </summary>
38          public readonly TLink Index;
39          /// <summary>
40          /// <para>
```

```
        /// The source.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly TLink Source;
        /// <summary>
        /// <para>
        /// The target.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly TLink Target;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Link"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="values">
        /// <para>A values.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(params TLink[] values) => SetValues(values, out Index, out Source, out
        ↪  Target);

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Link"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="values">
        /// <para>A values.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(IList<TLink> values) => SetValues(values, out Index, out Source, out Target);

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Link"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
        /// <para>A other.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="NotSupportedException">
        /// <para></para>
        /// <para></para>
        /// </exception>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(object other)
        {
            if (other is Link<TLink> otherLink)
            {
                SetValues(ref otherLink, out Index, out Source, out Target);
            }
            else if(other is IList<TLink> otherList)
            {
                SetValues(otherList, out Index, out Source, out Target);
            }
            else
            {
                throw new NotSupportedException();
            }
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Link"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
```

```
        /// <para>A other.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(ref Link<TLink> other) => SetValues(ref other, out Index, out Source, out
        ↪  Target);

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Link"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="index">
        /// <para>A index.</para>
        /// <para></para>
        /// </param>
        /// <param name="source">
        /// <para>A source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>A target.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(TLink index, TLink source, TLink target)
        {
            Index = index;
            Source = source;
            Target = target;
        }

        /// <summary>
        /// <para>
        /// Sets the values using the specified other.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
        /// <para>The other.</para>
        /// <para></para>
        /// </param>
        /// <param name="index">
        /// <para>The index.</para>
        /// <para></para>
        /// </param>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private static void SetValues(ref Link<TLink> other, out TLink index, out TLink source,
        ↪  out TLink target)
        {
            index = other.Index;
            source = other.Source;
            target = other.Target;
        }

        /// <summary>
        /// <para>
        /// Sets the values using the specified values.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="values">
        /// <para>The values.</para>
        /// <para></para>
        /// </param>
        /// <param name="index">
        /// <para>The index.</para>
        /// <para></para>
        /// </param>
```

```
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private static void SetValues(IList<TLink> values, out TLink index, out TLink source,
        ↪  out TLink target)
        {
            switch (values.Count)
            {
                case 3:
                    index = values[0];
                    source = values[1];
                    target = values[2];
                    break;
                case 2:
                    index = values[0];
                    source = values[1];
                    target = default;
                    break;
                case 1:
                    index = values[0];
                    source = default;
                    target = default;
                    break;
                default:
                    index = default;
                    source = default;
                    target = default;
                    break;
            }
        }

        /// <summary>
        /// <para>
        /// Gets the hash code.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override int GetHashCode() => (Index, Source, Target).GetHashCode();

        /// <summary>
        /// <para>
        /// Determines whether this instance is null.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool IsNull() => _equalityComparer.Equals(Index, _constants.Null)
                             && _equalityComparer.Equals(Source, _constants.Null)
                             && _equalityComparer.Equals(Target, _constants.Null);

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
        /// <para>The other.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
```

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

```
        /// <para>
        /// Returns the string.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
        ↪  ToString(Source, Target) : ToString(Index, Source, Target);

        #region IList

        /// <summary>
        /// <para>
        /// Gets the count value.
        /// </para>
        /// <para></para>
        /// </summary>
        public int Count
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => Length;
        }

        /// <summary>
        /// <para>
        /// Gets the is read only value.
        /// </para>
        /// <para></para>
        /// </summary>
        public bool IsReadOnly
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => true;
        }

        /// <summary>
        /// <para>
        /// The not supported exception.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink this[int index]
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get
            {
                Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
                ↪  nameof(index));
                if (index == _constants.IndexPart)
                {
                    return Index;
                }
                if (index == _constants.SourcePart)
                {
                    return Source;
                }
                if (index == _constants.TargetPart)
                {
                    return Target;
                }
                throw new NotSupportedException(); // Impossible path due to
                ↪  Ensure.ArgumentInRange
            }
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set => throw new NotSupportedException();
        }

        /// <summary>
        /// <para>
        /// Gets the enumerator.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The enumerator</para>
```

```
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();

        /// <summary>
        /// <para>
        /// Gets the enumerator.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>An enumerator of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public IEnumerator<TLink> GetEnumerator()
        {
            yield return Index;
            yield return Source;
            yield return Target;
        }

        /// <summary>
        /// <para>
        /// Adds the item.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="item">
        /// <para>The item.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Add(TLink item) => throw new NotSupportedException();

        /// <summary>
        /// <para>
        /// Clears this instance.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Clear() => throw new NotSupportedException();

        /// <summary>
        /// <para>
        /// Determines whether this instance contains.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="item">
        /// <para>The item.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Contains(TLink item) => IndexOf(item) >= 0;

        /// <summary>
        /// <para>
        /// Copies the to using the specified array.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="array">
        /// <para>The array.</para>
        /// <para></para>
        /// </param>
        /// <param name="arrayIndex">
        /// <para>The array index.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="InvalidOperationException">
        /// <para></para>
```

```
        /// <para></para>
        /// </exception>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void CopyTo(TLink[] array, int arrayIndex)
        {
            Ensure.OnDebug.ArgumentNotNull(array, nameof(array));
            Ensure.OnDebug.ArgumentInRange(arrayIndex, new Range<int>(0, array.Length - 1),
            ↪  nameof(arrayIndex));
            if (arrayIndex + Length > array.Length)
            {
                throw new InvalidOperationException();
            }
            array[arrayIndex++] = Index;
            array[arrayIndex++] = Source;
            array[arrayIndex] = Target;
        }

        /// <summary>
        /// <para>
        /// Determines whether this instance remove.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="item">
        /// <para>The item.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Remove(TLink item) => Throw.A.NotSupportedExceptionAndReturn<bool>();

        /// <summary>
        /// <para>
        /// Indexes the of using the specified item.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="item">
        /// <para>The item.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public int IndexOf(TLink item)
        {
            if (_equalityComparer.Equals(Index, item))
            {
                return _constants.IndexPart;
            }
            if (_equalityComparer.Equals(Source, item))
            {
                return _constants.SourcePart;
            }
            if (_equalityComparer.Equals(Target, item))
            {
                return _constants.TargetPart;
            }
            return -1;
        }

        /// <summary>
        /// <para>
        /// Inserts the index.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="index">
        /// <para>The index.</para>
        /// <para></para>
        /// </param>
        /// <param name="item">
        /// <para>The item.</para>
```

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

1.23 ./csharp/Platform.Data.Doublets/LinkExtensions.cs

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

```
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool IsPartialPoint<TLink>(this Link<TLink> link) =>
        ↪  Point<TLink>.IsPartialPoint(link);
    }
}
```

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets
{
    /// <summary>
    /// <para>
    /// Represents the links operator base.
    /// </para>
    /// <para></para>
    /// </summary>
    public abstract class LinksOperatorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly ILinks<TLink> _links;

        /// <summary>
        /// <para>
        /// Gets the links value.
        /// </para>
        /// <para></para>
        /// </summary>
        public ILinks<TLink> Links
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _links;
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksOperatorBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected LinksOperatorBase(ILinks<TLink> links) => _links = links;
    }
}
```

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory
{
    /// <summary>
    /// <para>
    /// Defines the links list methods.
    /// </para>
    /// <para></para>
    /// </summary>
    public interface ILinksListMethods<TLink>
    {
        /// <summary>
```

```
        /// <para>
        /// Detaches the free link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="freeLink">
        /// <para>The free link.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        void Detach(TLink freeLink);

        /// <summary>
        /// <para>
        /// Attaches the as first using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        void AttachAsFirst(TLink link);
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory
{
    /// <summary>
    /// <para>
    /// Defines the links tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    public interface ILinksTreeMethods<TLink>
    {
        /// <summary>
        /// <para>
        /// Counts the usages using the specified root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="root">
        /// <para>The root.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        TLink CountUsages(TLink root);

        /// <summary>
        /// <para>
        /// Searches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
```

```
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        TLink Search(TLink source, TLink target);

        /// <summary>
        /// <para>
        /// Eaches the usage using the specified root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="root">
        /// <para>The root.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        TLink EachUsage(TLink root, Func<IList<TLink>, TLink> handler);

        /// <summary>
        /// <para>
        /// Detaches the root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="root">
        /// <para>The root.</para>
        /// <para></para>
        /// </param>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        void Detach(ref TLink root, TLink linkIndex);

        /// <summary>
        /// <para>
        /// Attaches the root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="root">
        /// <para>The root.</para>
        /// <para></para>
        /// </param>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        void Attach(ref TLink root, TLink linkIndex);
    }
}
```

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

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

namespace Platform.Data.Doublets.Memory
{
    /// <summary>
    /// <para>
    /// The index tree type enum.
    /// </para>
    /// <para></para>
    /// </summary>
    public enum IndexTreeType
    {
        /// <summary>
        /// <para>
        /// The default index tree type.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        Default = 0,
        /// <summary>
        /// <para>
        /// The size balanced tree index tree type.
        /// </para>
        /// <para></para>
        /// </summary>
        SizeBalancedTree = 1,
        /// <summary>
        /// <para>
        /// The recursionless size balanced tree index tree type.
        /// </para>
        /// <para></para>
        /// </summary>
        RecursionlessSizeBalancedTree = 2,
        /// <summary>
        /// <para>
        /// The sized and threaded avl balanced tree index tree type.
        /// </para>
        /// <para></para>
        /// </summary>
        SizedAndThreadedAVLBalancedTree = 3
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Unsafe;

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

namespace Platform.Data.Doublets.Memory
{
    /// <summary>
    /// <para>
    /// The links header.
    /// </para>
    /// <para></para>
    /// </summary>
    public struct LinksHeader<TLink> : IEquatable<LinksHeader<TLink>>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// The size.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly long SizeInBytes = Structure<LinksHeader<TLink>>.Size;

        /// <summary>
        /// <para>
        /// The allocated links.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink AllocatedLinks;
        /// <summary>
        /// <para>
        /// The reserved links.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink ReservedLinks;
        /// <summary>
        /// <para>
        /// The free links.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink FreeLinks;
        /// <summary>
        /// <para>
```

```
        /// The first free link.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink FirstFreeLink;
        /// <summary>
        /// <para>
        /// The root as source.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink RootAsSource;
        /// <summary>
        /// <para>
        /// The root as target.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink RootAsTarget;
        /// <summary>
        /// <para>
        /// The last free link.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink LastFreeLink;
        /// <summary>
        /// <para>
        /// The reserved.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink Reserved8;

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="obj">
        /// <para>The obj.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override bool Equals(object obj) => obj is LinksHeader<TLink> linksHeader ?
        ↪  Equals(linksHeader) : false;

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
        /// <para>The other.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Equals(LinksHeader<TLink> other)
            => _equalityComparer.Equals(AllocatedLinks, other.AllocatedLinks)
            && _equalityComparer.Equals(ReservedLinks, other.ReservedLinks)
            && _equalityComparer.Equals(FreeLinks, other.FreeLinks)
            && _equalityComparer.Equals(FirstFreeLink, other.FirstFreeLink)
            && _equalityComparer.Equals(RootAsSource, other.RootAsSource)
            && _equalityComparer.Equals(RootAsTarget, other.RootAsTarget)
            && _equalityComparer.Equals(LastFreeLink, other.LastFreeLink)
            && _equalityComparer.Equals(Reserved8, other.Reserved8);

        /// <summary>
```

```
        /// <para>
        /// Gets the hash code.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override int GetHashCode() => (AllocatedLinks, ReservedLinks, FreeLinks,
        ↪  FirstFreeLink, RootAsSource, RootAsTarget, LastFreeLink, Reserved8).GetHashCode();

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

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

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

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Methods.Trees;
using Platform.Converters;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the external links recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
    ↪  RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
    {
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;

        /// <summary>
        /// <para>
        /// The break.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Break;
        /// <summary>
        /// <para>
        /// The continue.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Continue;
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksIndexParts;
        /// <summary>
```

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

```
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <param name="rootSource">
        /// <para>The root source.</para>
        /// <para></para>
        /// </param>
        /// <param name="rootTarget">
        /// <para>The root target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
        ↪  rootSource, TLink rootTarget);

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <param name="rootSource">
        /// <para>The root source.</para>
        /// <para></para>
        /// </param>
        /// <param name="rootTarget">
        /// <para>The root target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
        ↪  rootSource, TLink rootTarget);

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪  AsRef<LinksHeader<TLink>>(Header);

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

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

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

```
                if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key < root.Key
                {
                    root = GetLeftOrDefault(root);
                }
                else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key > root.Key
                {
                    root = GetRightOrDefault(root);
                }
                else // node.Key == root.Key
                {
                    return root;
                }
            }
            return Zero;
        }

        // TODO: Return indices range instead of references count
        /// <summary>
        /// <para>
        /// Counts the usages using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink CountUsages(TLink link)
        {
            var root = GetTreeRoot();
            var total = GetSize(root);
            var totalRightIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (LessOrEqualThan(@base, link))
                {
                    root = GetRightOrDefault(root);
                }
                else
                {
                    totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
                    root = GetLeftOrDefault(root);
                }
            }
            root = GetTreeRoot();
            var totalLeftIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (GreaterOrEqualThan(@base, link))
                {
                    root = GetLeftOrDefault(root);
                }
                else
                {
                    totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
                    root = GetRightOrDefault(root);
                }
            }
            return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
        }

        /// <summary>
        /// <para>
        /// Eaches the usage using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
```

```
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
        ↪  EachUsageCore(@base, GetTreeRoot(), handler);

        // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
        ↪  low-level MSIL stack.
        /// <summary>
        /// <para>
        /// Eaches the usage core using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The continue.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
        {
            var @continue = Continue;
            if (EqualToZero(link))
            {
                return @continue;
            }
            var linkBasePart = GetBasePartValue(link);
            var @break = Break;
            if (GreaterThan(linkBasePart, @base))
            {
                if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            else if (LessThan(linkBasePart, @base))
            {
                if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            else //if (linkBasePart == @base)
            {
                if (AreEqual(handler(GetLinkValues(link)), @break))
                {
                    return @break;
                }
                if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
                {
                    return @break;
                }
                if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            return @continue;
```

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

1.30 ./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> :
       ↪  SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
22     {
23         private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
           ↪  UncheckedConverter<TLink, long>.Default;
24
25         /// <summary>
26         /// <para>
27         /// The break.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         protected readonly TLink Break;
32         /// <summary>
33         /// <para>
34         /// The continue.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         protected readonly TLink Continue;
39         /// <summary>
40         /// <para>
41         /// The links data parts.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         protected readonly byte* LinksDataParts;
46         /// <summary>
47         /// <para>
```

```
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksIndexParts;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="ExternalLinksSizeBalancedTreeMethodsBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
        ↪  byte* linksDataParts, byte* linksIndexParts, byte* header)
        {
            LinksDataParts = linksDataParts;
            LinksIndexParts = linksIndexParts;
            Header = header;
            Break = constants.Break;
            Continue = constants.Continue;
        }

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract TLink GetTreeRoot();

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

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

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

```
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↪  AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
        ↪  _addressToInt64Converter.Convert(link)));

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪  ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
        ↪  (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));

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

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
        {
            ref var firstLink = ref GetLinkDataPartReference(first);
            ref var secondLink = ref GetLinkDataPartReference(second);
            return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
            ↪  secondLink.Source, secondLink.Target);
```

```
273         }
274
275         /// <summary>
276         /// <para>
277         /// Determines whether this instance first is to the right of second.
278         /// </para>
279         /// <para></para>
280         /// </summary>
281         /// <param name="first">
282         /// <para>The first.</para>
283         /// <para></para>
284         /// </param>
285         /// <param name="second">
286         /// <para>The second.</para>
287         /// <para></para>
288         /// </param>
289         /// <returns>
290         /// <para>The bool</para>
291         /// <para></para>
292         /// </returns>
293         [MethodImpl(MethodImplOptions.AggressiveInlining)]
294         protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
295         {
296             ref var firstLink = ref GetLinkDataPartReference(first);
297             ref var secondLink = ref GetLinkDataPartReference(second);
298             return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
                ↪  secondLink.Source, secondLink.Target);
299         }
300
301         /// <summary>
302         /// <para>
303         /// The zero.
304         /// </para>
305         /// <para></para>
306         /// </summary>
307         public TLink this[TLink index]
308         {
309             [MethodImpl(MethodImplOptions.AggressiveInlining)]
310             get
311             {
312                 var root = GetTreeRoot();
313                 if (GreaterOrEqualThan(index, GetSize(root)))
314                 {
315                     return Zero;
316                 }
317                 while (!EqualToZero(root))
318                 {
319                     var left = GetLeftOrDefault(root);
320                     var leftSize = GetSizeOrZero(left);
321                     if (LessThan(index, leftSize))
322                     {
323                         root = left;
324                         continue;
325                     }
326                     if (AreEqual(index, leftSize))
327                     {
328                         return root;
329                     }
330                     root = GetRightOrDefault(root);
331                     index = Subtract(index, Increment(leftSize));
332                 }
333                 return Zero; // TODO: Impossible situation exception (only if tree structure
                    ↪  broken)
334             }
335         }
336
337         /// <summary>
338         /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
            ↪  (концом).
339         /// </summary>
340         /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
341         /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
342         /// <returns>Индекс искомой связи.</returns>
343         [MethodImpl(MethodImplOptions.AggressiveInlining)]
344         public TLink Search(TLink source, TLink target)
345         {
346             var root = GetTreeRoot();
347             while (!EqualToZero(root))
```

```
            {
                ref var rootLink = ref GetLinkDataPartReference(root);
                var rootSource = rootLink.Source;
                var rootTarget = rootLink.Target;
                if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key < root.Key
                {
                    root = GetLeftOrDefault(root);
                }
                else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key > root.Key
                {
                    root = GetRightOrDefault(root);
                }
                else // node.Key == root.Key
                {
                    return root;
                }
            }
            return Zero;
        }

        // TODO: Return indices range instead of references count
        /// <summary>
        /// <para>
        /// Counts the usages using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink CountUsages(TLink link)
        {
            var root = GetTreeRoot();
            var total = GetSize(root);
            var totalRightIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (LessOrEqualThan(@base, link))
                {
                    root = GetRightOrDefault(root);
                }
                else
                {
                    totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
                    root = GetLeftOrDefault(root);
                }
            }
            root = GetTreeRoot();
            var totalLeftIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (GreaterOrEqualThan(@base, link))
                {
                    root = GetLeftOrDefault(root);
                }
                else
                {
                    totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
                    root = GetRightOrDefault(root);
                }
            }
            return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
        }

        /// <summary>
        /// <para>
        /// Eaches the usage using the specified base.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
        ↪  EachUsageCore(@base, GetTreeRoot(), handler);

        // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
        ↪  low-level MSIL stack.
        /// <summary>
        /// <para>
        /// Eaches the usage core using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The continue.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
        {
            var @continue = Continue;
            if (EqualToZero(link))
            {
                return @continue;
            }
            var linkBasePart = GetBasePartValue(link);
            var @break = Break;
            if (GreaterThan(linkBasePart, @base))
            {
                if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            else if (LessThan(linkBasePart, @base))
            {
                if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            else //if (linkBasePart == @base)
            {
                if (AreEqual(handler(GetLinkValues(link)), @break))
                {
                    return @break;
                }
                if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
                {
                    return @break;
                }
                if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
                {
```

```
                    return @break;
                }
            }
            return @continue;
        }

        /// <summary>
        /// <para>
        /// Prints the node value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="sb">
        /// <para>The sb.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void PrintNodeValue(TLink node, StringBuilder sb)
        {
            ref var link = ref GetLinkDataPartReference(node);
            sb.Append(' ');
            sb.Append(link.Source);
            sb.Append('-');
            sb.Append('>');
            sb.Append(link.Target);
        }
    }
}
```

1.31 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSourcesRecursionlessSizeBalancedTree

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the external links sources recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe class ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink> :
    ↪  ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪  constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
        ↪  base(constants, linksDataParts, linksIndexParts, header) { }

```

```
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) =>
                ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
125
126             /// <summary>
127             /// <para>
128             /// Sets the right using the specified node.
129             /// </para>
130             /// <para></para>
131             /// </summary>
132             /// <param name="node">
133             /// <para>The node.</para>
134             /// <para></para>
135             /// </param>
136             /// <param name="right">
137             /// <para>The right.</para>
138             /// <para></para>
139             /// </param>
140             [MethodImpl(MethodImplOptions.AggressiveInlining)]
141             protected override void SetRight(TLink node, TLink right) =>
                ↪ GetLinkIndexPartReference(node).RightAsSource = right;
142
143             /// <summary>
144             /// <para>
145             /// Gets the size using the specified node.
146             /// </para>
147             /// <para></para>
148             /// </summary>
149             /// <param name="node">
150             /// <para>The node.</para>
151             /// <para></para>
152             /// </param>
153             /// <returns>
154             /// <para>The link</para>
155             /// <para></para>
156             /// </returns>
157             [MethodImpl(MethodImplOptions.AggressiveInlining)]
158             protected override TLink GetSize(TLink node) =>
                ↪ GetLinkIndexPartReference(node).SizeAsSource;
159
160             /// <summary>
161             /// <para>
162             /// Sets the size using the specified node.
163             /// </para>
164             /// <para></para>
165             /// </summary>
166             /// <param name="node">
167             /// <para>The node.</para>
168             /// <para></para>
169             /// </param>
170             /// <param name="size">
171             /// <para>The size.</para>
172             /// <para></para>
173             /// </param>
174             [MethodImpl(MethodImplOptions.AggressiveInlining)]
175             protected override void SetSize(TLink node, TLink size) =>
                ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
176
177             /// <summary>
178             /// <para>
179             /// Gets the tree root.
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) =>
            ↪ GetLinkDataPartReference(link).Source;
206
207         /// <summary>
208         /// <para>
209         /// Determines whether this instance first is to the left of second.
210         /// </para>
211         /// <para></para>
212         /// </summary>
213         /// <param name="firstSource">
214         /// <para>The first source.</para>
215         /// <para></para>
216         /// </param>
217         /// <param name="firstTarget">
218         /// <para>The first target.</para>
219         /// <para></para>
220         /// </param>
221         /// <param name="secondSource">
222         /// <para>The second source.</para>
223         /// <para></para>
224         /// </param>
225         /// <param name="secondTarget">
226         /// <para>The second target.</para>
227         /// <para></para>
228         /// </param>
229         /// <returns>
230         /// <para>The bool</para>
231         /// <para></para>
232         /// </returns>
233         [MethodImpl(MethodImplOptions.AggressiveInlining)]
234         protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
            ↪ TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
            ↪ (AreEqual(firstSource, secondSource) && LessThan(firstTarget, secondTarget));
235
236         /// <summary>
237         /// <para>
238         /// Determines whether this instance first is to the right of second.
239         /// </para>
240         /// <para></para>
241         /// </summary>
242         /// <param name="firstSource">
243         /// <para>The first source.</para>
244         /// <para></para>
245         /// </param>
246         /// <param name="firstTarget">
247         /// <para>The first target.</para>
248         /// <para></para>
249         /// </param>
250         /// <param name="secondSource">
251         /// <para>The second source.</para>
252         /// <para></para>
253         /// </param>
254         /// <param name="secondTarget">
255         /// <para>The second target.</para>
256         /// <para></para>
257         /// </param>
258         /// <returns>
259         /// <para>The bool</para>
260         /// <para></para>
261         /// </returns>
262         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

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

1.32 ./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) =>
            ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
125
126         /// <summary>
127         /// <para>
128         /// Sets the right using the specified node.
129         /// </para>
130         /// <para></para>
131         /// </summary>
132         /// <param name="node">
133         /// <para>The node.</para>
134         /// <para></para>
135         /// </param>
136         /// <param name="right">
137         /// <para>The right.</para>
138         /// <para></para>
139         /// </param>
140         [MethodImpl(MethodImplOptions.AggressiveInlining)]
141         protected override void SetRight(TLink node, TLink right) =>
            ↪ GetLinkIndexPartReference(node).RightAsSource = right;
142
143         /// <summary>
144         /// <para>
145         /// Gets the size using the specified node.
146         /// </para>
147         /// <para></para>
148         /// </summary>
149         /// <param name="node">
150         /// <para>The node.</para>
151         /// <para></para>
152         /// </param>
153         /// <returns>
154         /// <para>The link</para>
155         /// <para></para>
156         /// </returns>
157         [MethodImpl(MethodImplOptions.AggressiveInlining)]
158         protected override TLink GetSize(TLink node) =>
            ↪ GetLinkIndexPartReference(node).SizeAsSource;
159
160         /// <summary>
161         /// <para>
162         /// Sets the size using the specified node.
163         /// </para>
164         /// <para></para>
165         /// </summary>
166         /// <param name="node">
167         /// <para>The node.</para>
168         /// <para></para>
169         /// </param>
170         /// <param name="size">
171         /// <para>The size.</para>
172         /// <para></para>
173         /// </param>
174         [MethodImpl(MethodImplOptions.AggressiveInlining)]
175         protected override void SetSize(TLink node, TLink size) =>
            ↪ GetLinkIndexPartReference(node).SizeAsSource = size;
176
177         /// <summary>
178         /// <para>
179         /// Gets the tree root.
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>
```

```
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetBasePartValue(TLink link) =>
        ↪  GetLinkDataPartReference(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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
        ↪  (AreEqual(firstSource, secondSource) && LessThan(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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
        ↪  (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
```

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

1.33 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksTargetsRecursionlessSizeBalancedTree

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the external links targets recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe class ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
    ↪  ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪  constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
        ↪  base(constants, linksDataParts, linksIndexParts, header) { }

        /// <summary>
        /// <para>
        /// Gets the left reference using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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;
125
126         /// <summary>
127         /// <para>
128         /// Sets the right using the specified node.
129         /// </para>
130         /// <para></para>
```

```
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  GetLinkIndexPartReference(node).RightAsTarget = right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetSize(TLink node) =>
        ↪  GetLinkIndexPartReference(node).SizeAsTarget;

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  GetLinkIndexPartReference(node).SizeAsTarget = size;

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

        /// <summary>
        /// <para>
        /// Gets the base part value using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

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

```
        {
            ref var link = ref GetLinkIndexPartReference(node);
            link.LeftAsTarget = Zero;
            link.RightAsTarget = Zero;
            link.SizeAsTarget = Zero;
        }
    }
}
```

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the external links targets size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe class ExternalLinksTargetsSizeBalancedTreeMethods<TLink> :
    ↪ ExternalLinksSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="ExternalLinksTargetsSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
        ↪ byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
        ↪ linksDataParts, linksIndexParts, header) { }

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

        /// <summary>
        /// <para>
        /// Gets the right reference using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <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;
125
126         /// <summary>
127         /// <para>
128         /// Sets the right using the specified node.
129         /// </para>
130         /// <para></para>
131         /// </summary>
132         /// <param name="node">
133         /// <para>The node.</para>
134         /// <para></para>
135         /// </param>
136         /// <param name="right">
137         /// <para>The right.</para>
138         /// <para></para>
```

```
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪ GetLinkIndexPartReference(node).RightAsTarget = right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetSize(TLink node) =>
        ↪ GetLinkIndexPartReference(node).SizeAsTarget;

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪ GetLinkIndexPartReference(node).SizeAsTarget = size;

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

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

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
```

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

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Methods.Trees;
using Platform.Converters;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the internal links recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
    ↪ RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
    {
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪ UncheckedConverter<TLink, long>.Default;

        /// <summary>
        /// <para>
        /// The break.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Break;
        /// <summary>
        /// <para>
        /// The continue.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Continue;
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksIndexParts;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪ cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
```

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

```
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↪  AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
        ↪  _addressToInt64Converter.Convert(link)));

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪  ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
        ↪  (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));

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

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

        /// <summary>
        /// <para>
        /// Gets the link values using the specified link index.
        /// </para>
        /// <para></para>
```

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

```
        /// <para>The zero.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected TLink SearchCore(TLink root, TLink key)
        {
            while (!EqualToZero(root))
            {
                var rootKey = GetKeyPartValue(root);
                if (LessThan(key, rootKey)) // node.Key < root.Key
                {
                    root = GetLeftOrDefault(root);
                }
                else if (GreaterThan(key, rootKey)) // node.Key > root.Key
                {
                    root = GetRightOrDefault(root);
                }
                else // node.Key == root.Key
                {
                    return root;
                }
            }
            return Zero;
        }

        // TODO: Return indices range instead of references count
        /// <summary>
        /// <para>
        /// Counts the usages using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink CountUsages(TLink link) => GetSizeOrZero(GetTreeRoot(link));

        /// <summary>
        /// <para>
        /// Eaches the usage using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
        ↪  EachUsageCore(@base, GetTreeRoot(@base), handler);

        // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
        ↪  low-level MSIL stack.
        /// <summary>
        /// <para>
        /// Eaches the usage core using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
        /// <param name="link">
```

```
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The continue.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
        {
            var @continue = Continue;
            if (EqualToZero(link))
            {
                return @continue;
            }
            var @break = Break;
            if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
            {
                return @break;
            }
            if (AreEqual(handler(GetLinkValues(link)), @break))
            {
                return @break;
            }
            if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
            {
                return @break;
            }
            return @continue;
        }

        /// <summary>
        /// <para>
        /// Prints the node value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="sb">
        /// <para>The sb.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void PrintNodeValue(TLink node, StringBuilder sb)
        {
            ref var link = ref GetLinkDataPartReference(node);
            sb.Append(' ');
            sb.Append(link.Source);
            sb.Append('-');
            sb.Append('>');
            sb.Append(link.Target);
        }
    }
}
```

1.36 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSizeBalancedTreeMethodsBase.cs

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Methods.Trees;
using Platform.Converters;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the internal links size balanced tree methods base.
    /// </para>
```

```
    /// <para></para>
    /// </summary>
    /// <seealso cref="SizeBalancedTreeMethods{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class InternalLinksSizeBalancedTreeMethodsBase<TLink> :
    ↪ SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
    {
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪ UncheckedConverter<TLink, long>.Default;

        /// <summary>
        /// <para>
        /// The break.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Break;
        /// <summary>
        /// <para>
        /// The continue.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Continue;
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* LinksIndexParts;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="InternalLinksSizeBalancedTreeMethodsBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
        ↪ byte* linksDataParts, byte* linksIndexParts, byte* header)
        {
            LinksDataParts = linksDataParts;
            LinksIndexParts = linksIndexParts;
            Header = header;
            Break = constants.Break;
            Continue = constants.Continue;
        }

```

```
        /// <summary>
        /// <para>
        /// Gets the tree root using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract TLink GetTreeRoot(TLink link);

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

        /// <summary>
        /// <para>
        /// Gets the key part value using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract TLink GetKeyPartValue(TLink link);

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↪  AsRef<RawLinkDataPart<TLink>>(LinksDataParts + (RawLinkDataPart<TLink>.SizeInBytes *
        ↪  _addressToInt64Converter.Convert(link)));

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
```

```
169         /// <para></para>
170         /// </param>
171         /// <returns>
172         /// <para>A ref raw link index part of t link</para>
173         /// <para></para>
174         /// </returns>
175         [MethodImpl(MethodImplOptions.AggressiveInlining)]
176         protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
            ↪ ref AsRef<RawLinkIndexPart<TLink>>(LinksIndexParts +
            ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
177
178         /// <summary>
179         /// <para>
180         /// Determines whether this instance first is to the left of second.
181         /// </para>
182         /// <para></para>
183         /// </summary>
184         /// <param name="first">
185         /// <para>The first.</para>
186         /// <para></para>
187         /// </param>
188         /// <param name="second">
189         /// <para>The second.</para>
190         /// <para></para>
191         /// </param>
192         /// <returns>
193         /// <para>The bool</para>
194         /// <para></para>
195         /// </returns>
196         [MethodImpl(MethodImplOptions.AggressiveInlining)]
197         protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second) =>
            ↪ LessThan(GetKeyPartValue(first), GetKeyPartValue(second));
198
199         /// <summary>
200         /// <para>
201         /// Determines whether this instance first is to the right of second.
202         /// </para>
203         /// <para></para>
204         /// </summary>
205         /// <param name="first">
206         /// <para>The first.</para>
207         /// <para></para>
208         /// </param>
209         /// <param name="second">
210         /// <para>The second.</para>
211         /// <para></para>
212         /// </param>
213         /// <returns>
214         /// <para>The bool</para>
215         /// <para></para>
216         /// </returns>
217         [MethodImpl(MethodImplOptions.AggressiveInlining)]
218         protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second) =>
            ↪ GreaterThan(GetKeyPartValue(first), GetKeyPartValue(second));
219
220         /// <summary>
221         /// <para>
222         /// Gets the link values using the specified link index.
223         /// </para>
224         /// <para></para>
225         /// </summary>
226         /// <param name="linkIndex">
227         /// <para>The link index.</para>
228         /// <para></para>
229         /// </param>
230         /// <returns>
231         /// <para>A list of t link</para>
232         /// <para></para>
233         /// </returns>
234         [MethodImpl(MethodImplOptions.AggressiveInlining)]
235         protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
236         {
237             ref var link = ref GetLinkDataPartReference(linkIndex);
238             return new Link<TLink>(linkIndex, link.Source, link.Target);
239         }
240
241         /// <summary>
242         /// <para>
```

```
        /// The zero.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink this[TLink link, TLink index]
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get
            {
                var root = GetTreeRoot(link);
                if (GreaterOrEqualThan(index, GetSize(root)))
                {
                    return Zero;
                }
                while (!EqualToZero(root))
                {
                    var left = GetLeftOrDefault(root);
                    var leftSize = GetSizeOrZero(left);
                    if (LessThan(index, leftSize))
                    {
                        root = left;
                        continue;
                    }
                    if (AreEqual(index, leftSize))
                    {
                        return root;
                    }
                    root = GetRightOrDefault(root);
                    index = Subtract(index, Increment(leftSize));
                }
                return Zero; // TODO: Impossible situation exception (only if tree structure
                ↪  broken)
            }
        }

        /// <summary>
        /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
        ↪  (концом).
        /// </summary>
        /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
        /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
        /// <returns>Индекс искомой связи.</returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public abstract TLink Search(TLink source, TLink target);

        /// <summary>
        /// <para>
        /// Searches the core using the specified root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="root">
        /// <para>The root.</para>
        /// <para></para>
        /// </param>
        /// <param name="key">
        /// <para>The key.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The zero.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected TLink SearchCore(TLink root, TLink key)
        {
            while (!EqualToZero(root))
            {
                var rootKey = GetKeyPartValue(root);
                if (LessThan(key, rootKey)) // node.Key < root.Key
                {
                    root = GetLeftOrDefault(root);
                }
                else if (GreaterThan(key, rootKey)) // node.Key > root.Key
                {
                    root = GetRightOrDefault(root);
                }
                else // node.Key == root.Key
```

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

```
            }
            var @break = Break;
            if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
            {
                return @break;
            }
            if (AreEqual(handler(GetLinkValues(link)), @break))
            {
                return @break;
            }
            if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
            {
                return @break;
            }
            return @continue;
        }

        /// <summary>
        /// <para>
        /// Prints the node value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="sb">
        /// <para>The sb.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void PrintNodeValue(TLink node, StringBuilder sb)
        {
            ref var link = ref GetLinkDataPartReference(node);
            sb.Append(' ');
            sb.Append(link.Source);
            sb.Append('-');
            sb.Append('>');
            sb.Append(link.Target);
        }
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Methods.Lists;
using Platform.Converters;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the internal links sources linked list methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="RelativeCircularDoublyLinkedListMethods{TLink}"/>
    public unsafe class InternalLinksSourcesLinkedListMethods<TLink> :
    ↪  RelativeCircularDoublyLinkedListMethods<TLink>
    {
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;
        private readonly byte* _linksDataParts;
        private readonly byte* _linksIndexParts;
        /// <summary>
        /// <para>
        /// The break.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Break;
        /// <summary>
        /// <para>
```

```
        /// The continue.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Continue;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="InternalLinksSourcesLinkedListMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants, byte*
        ↪  linksDataParts, byte* linksIndexParts)
        {
            _linksDataParts = linksDataParts;
            _linksIndexParts = linksIndexParts;
            Break = constants.Break;
            Continue = constants.Continue;
        }

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↪  AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (RawLinkDataPart<TLink>.SizeInBytes
        ↪  * _addressToInt64Converter.Convert(link)));

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪  ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
        ↪  (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));

        /// <summary>
        /// <para>
        /// Gets the first using the specified head.
        /// </para>
        /// <para></para>
        /// </summary>
```

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

```
        /// Gets the size using the specified head.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="head">
        /// <para>The head.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetSize(TLink head) =>
        ↪  GetLinkIndexPartReference(head).SizeAsSource;

        /// <summary>
        /// <para>
        /// Sets the first using the specified head.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="head">
        /// <para>The head.</para>
        /// <para></para>
        /// </param>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetFirst(TLink head, TLink element) =>
        ↪  GetLinkIndexPartReference(head).RootAsSource = element;

        /// <summary>
        /// <para>
        /// Sets the last using the specified head.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="head">
        /// <para>The head.</para>
        /// <para></para>
        /// </param>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLast(TLink head, TLink element)
        {
            //var first = GetLinkIndexPartReference(head).RootAsSource;
            //if (EqualToZero(first))
            //{
            //    SetFirst(head, element);
            //}
            //else
            //{
            //    SetPrevious(first, element);
            //}
        }

        /// <summary>
        /// <para>
        /// Sets the previous using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        /// <param name="previous">
        /// <para>The previous.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

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

```
        /// <para>
        /// Eaches the usage using the specified source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The continue.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink EachUsage(TLink source, Func<IList<TLink>, TLink> handler)
        {
            var @continue = Continue;
            var @break = Break;
            var current = GetFirst(source);
            var first = current;
            while (!EqualToZero(current))
            {
                if (AreEqual(handler(GetLinkValues(current)), @break))
                {
                    return @break;
                }
                current = GetNext(current);
                if (AreEqual(current, first))
                {
                    return @continue;
                }
            }
            return @continue;
        }
    }
}
```

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the internal links sources recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe class InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink> :
    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪ cref="InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
```

```
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
        ↪  constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
        ↪  base(constants, linksDataParts, linksIndexParts, header) { }

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

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

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

        /// <summary>
        /// <para>
        /// Gets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

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

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

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

1.39 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksSourcesSizeBalancedTreeMethods.cs

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

```
58          /// <summary>
59          /// <para>
60          /// Gets the right reference using the specified node.
61          /// </para>
62          /// <para></para>
63          /// </summary>
64          /// <param name="node">
65          /// <para>The node.</para>
66          /// <para></para>
67          /// </param>
68          /// <returns>
69          /// <para>The ref link</para>
70          /// <para></para>
71          /// </returns>
72          [MethodImpl(MethodImplOptions.AggressiveInlining)]
73          protected override ref TLink GetRightReference(TLink node) => ref
            ↪ 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) =>
            ↪ GetLinkIndexPartReference(node).LeftAsSource = left;
125
126         /// <summary>
127         /// <para>
128         /// Sets the right using the specified node.
129         /// </para>
130         /// <para></para>
131         /// </summary>
```

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

```
206         /// <para></para>
207         /// </returns>
208         [MethodImpl(MethodImplOptions.AggressiveInlining)]
209         protected override TLink GetBasePartValue(TLink link) =>
            ↪ GetLinkDataPartReference(link).Source;
210
211         /// <summary>
212         /// <para>
213         /// Gets the key part value using the specified link.
214         /// </para>
215         /// <para></para>
216         /// </summary>
217         /// <param name="link">
218         /// <para>The link.</para>
219         /// <para></para>
220         /// </param>
221         /// <returns>
222         /// <para>The link</para>
223         /// <para></para>
224         /// </returns>
225         [MethodImpl(MethodImplOptions.AggressiveInlining)]
226         protected override TLink GetKeyPartValue(TLink link) =>
            ↪ GetLinkDataPartReference(link).Target;
227
228         /// <summary>
229         /// <para>
230         /// Clears the node using the specified node.
231         /// </para>
232         /// <para></para>
233         /// </summary>
234         /// <param name="node">
235         /// <para>The node.</para>
236         /// <para></para>
237         /// </param>
238         [MethodImpl(MethodImplOptions.AggressiveInlining)]
239         protected override void ClearNode(TLink node)
240         {
241             ref var link = ref GetLinkIndexPartReference(node);
242             link.LeftAsSource = Zero;
243             link.RightAsSource = Zero;
244             link.SizeAsSource = Zero;
245         }
246
247         /// <summary>
248         /// <para>
249         /// Searches the source.
250         /// </para>
251         /// <para></para>
252         /// </summary>
253         /// <param name="source">
254         /// <para>The source.</para>
255         /// <para></para>
256         /// </param>
257         /// <param name="target">
258         /// <para>The target.</para>
259         /// <para></para>
260         /// </param>
261         /// <returns>
262         /// <para>The link</para>
263         /// <para></para>
264         /// </returns>
265         public override TLink Search(TLink source, TLink target) =>
            ↪ 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
            ↪  cref="InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
19          /// </para>
20          /// <para></para>
21          /// </summary>
22          /// <param name="constants">
23          /// <para>A constants.</para>
24          /// <para></para>
25          /// </param>
26          /// <param name="linksDataParts">
27          /// <para>A links data parts.</para>
28          /// <para></para>
29          /// </param>
30          /// <param name="linksIndexParts">
31          /// <para>A links index parts.</para>
32          /// <para></para>
33          /// </param>
34          /// <param name="header">
35          /// <para>A header.</para>
36          /// <para></para>
37          /// </param>
38          [MethodImpl(MethodImplOptions.AggressiveInlining)]
39          public InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
            ↪  constants, byte* linksDataParts, byte* linksIndexParts, byte* header) :
            ↪  base(constants, linksDataParts, linksIndexParts, header) { }
40
41          /// <summary>
42          /// <para>
43          /// Gets the left reference using the specified node.
44          /// </para>
45          /// <para></para>
46          /// </summary>
47          /// <param name="node">
48          /// <para>The node.</para>
49          /// <para></para>
50          /// </param>
51          /// <returns>
52          /// <para>The ref link</para>
53          /// <para></para>
54          /// </returns>
55          [MethodImpl(MethodImplOptions.AggressiveInlining)]
56          protected override ref TLink GetLeftReference(TLink node) => ref
            ↪  GetLinkIndexPartReference(node).LeftAsTarget;
57
58          /// <summary>
59          /// <para>
60          /// Gets the right reference using the specified node.
61          /// </para>
62          /// <para></para>
63          /// </summary>
64          /// <param name="node">
65          /// <para>The node.</para>
66          /// <para></para>
67          /// </param>
68          /// <returns>
69          /// <para>The ref link</para>
70          /// <para></para>
71          /// </returns>
72          [MethodImpl(MethodImplOptions.AggressiveInlining)]
73          protected override ref TLink GetRightReference(TLink node) => ref
            ↪  GetLinkIndexPartReference(node).RightAsTarget;
74
75          /// <summary>
76          /// <para>
77          /// Gets the left using the specified node.
78          /// </para>
79          /// <para></para>
80          /// </summary>
81          /// <param name="node">
82          /// <para>The node.</para>
```

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

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

```
229         /// <para>
230         /// Clears the node using the specified node.
231         /// </para>
232         /// <para></para>
233         /// </summary>
234         /// <param name="node">
235         /// <para>The node.</para>
236         /// <para></para>
237         /// </param>
238         [MethodImpl(MethodImplOptions.AggressiveInlining)]
239         protected override void ClearNode(TLink node)
240         {
241             ref var link = ref GetLinkIndexPartReference(node);
242             link.LeftAsTarget = Zero;
243             link.RightAsTarget = Zero;
244             link.SizeAsTarget = Zero;
245         }
246 
247         /// <summary>
248         /// <para>
249         /// Searches the source.
250         /// </para>
251         /// <para></para>
252         /// </summary>
253         /// <param name="source">
254         /// <para>The source.</para>
255         /// <para></para>
256         /// </param>
257         /// <param name="target">
258         /// <para>The target.</para>
259         /// <para></para>
260         /// </param>
261         /// <returns>
262         /// <para>The link</para>
263         /// <para></para>
264         /// </returns>
265         public override TLink Search(TLink source, TLink target) =>
            ↪ SearchCore(GetTreeRoot(target), source);
266     }
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> :
        ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="InternalLinksTargetsSizeBalancedTreeMethods"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="constants">
23         /// <para>A constants.</para>
24         /// <para></para>
25         /// </param>
26         /// <param name="linksDataParts">
27         /// <para>A links data parts.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="linksIndexParts">
31         /// <para>A links index parts.</para>
32         /// <para></para>
33         /// </param>
34         /// <param name="header">
35         /// <para>A header.</para>
```

```
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants,
        ↪  byte* linksDataParts, byte* linksIndexParts, byte* header) : base(constants,
        ↪  linksDataParts, linksIndexParts, header) { }

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

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

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

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

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

```
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetTreeRoot(TLink link) =>
        ↪  GetLinkIndexPartReference(link).RootAsTarget;

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

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

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(TLink node)
        {
            ref var link = ref GetLinkIndexPartReference(node);
            link.LeftAsTarget = Zero;
            link.RightAsTarget = Zero;
            link.SizeAsTarget = Zero;
        }

        /// <summary>
        /// <para>
        /// Searches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
```

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

1.42 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinks.cs

```
1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using static System.Runtime.CompilerServices.Unsafe;
6
7  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9  namespace Platform.Data.Doublets.Memory.Split.Generic
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the split memory links.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
18     public unsafe class SplitMemoryLinks<TLink> : SplitMemoryLinksBase<TLink>
19     {
20         private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
21         private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
24         private byte* _header;
25         private byte* _linksDataParts;
26         private byte* _linksIndexParts;
27
28         /// <summary>
29         /// <para>
30         /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         /// <param name="dataMemory">
35         /// <para>A data memory.</para>
36         /// <para></para>
37         /// </param>
38         /// <param name="indexMemory">
39         /// <para>A index memory.</para>
40         /// <para></para>
41         /// </param>
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         public SplitMemoryLinks(string dataMemory, string indexMemory) : this(new
            ↪ FileMappedResizableDirectMemory(dataMemory), new
            ↪ FileMappedResizableDirectMemory(indexMemory)) { }
44
45         /// <summary>
46         /// <para>
47         /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         /// <param name="dataMemory">
52         /// <para>A data memory.</para>
53         /// <para></para>
54         /// </param>
55         /// <param name="indexMemory">
56         /// <para>A index memory.</para>
57         /// <para></para>
58         /// </param>
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
61
62         /// <summary>
```

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

```
        /// <param name="useLinkedList">
        /// <para>A use linked list.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪  indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
        ↪  IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↪  memoryReservationStep, constants, useLinkedList)
        {
            if (indexTreeType == IndexTreeType.SizeBalancedTree)
            {
                _createInternalSourceTreeMethods = () => new
                ↪  InternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createExternalSourceTreeMethods = () => new
                ↪  ExternalLinksSourcesSizeBalancedTreeMethods<TLink>(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createInternalTargetTreeMethods = () => new
                ↪  InternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createExternalTargetTreeMethods = () => new
                ↪  ExternalLinksTargetsSizeBalancedTreeMethods<TLink>(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
            }
            else
            {
                _createInternalSourceTreeMethods = () => new
                ↪  InternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createExternalSourceTreeMethods = () => new
                ↪  ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createInternalTargetTreeMethods = () => new
                ↪  InternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createExternalTargetTreeMethods = () => new
                ↪  ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods<TLink>(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
            }
            Init(dataMemory, indexMemory);
        }

        /// <summary>
        /// <para>
        /// Sets the pointers using the specified data memory.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="dataMemory">
        /// <para>The data memory.</para>
        /// <para></para>
        /// </param>
        /// <param name="indexMemory">
        /// <para>The index memory.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetPointers(IResizableDirectMemory dataMemory,
        ↪  IResizableDirectMemory indexMemory)
        {
            _linksDataParts = (byte*)dataMemory.Pointer;
            _linksIndexParts = (byte*)indexMemory.Pointer;
            _header = _linksIndexParts;
            if (_useLinkedList)
            {
                InternalSourcesListMethods = new
                ↪  InternalLinksSourcesLinkedListMethods<TLink>(Constants, _linksDataParts,
                ↪  _linksIndexParts);
            }
            else
            {
                InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
            }
            ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
            InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
            ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
```

```
189             UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_linksDataParts, _header);
190         }
191 
192         /// <summary>
193         /// <para>
194         /// Resets the pointers.
195         /// </para>
196         /// <para></para>
197         /// </summary>
198         [MethodImpl(MethodImplOptions.AggressiveInlining)]
199         protected override void ResetPointers()
200         {
201             base.ResetPointers();
202             _linksDataParts = null;
203             _linksIndexParts = null;
204             _header = null;
205         }
206 
207         /// <summary>
208         /// <para>
209         /// Gets the header reference.
210         /// </para>
211         /// <para></para>
212         /// </summary>
213         /// <returns>
214         /// <para>A ref links header of t link</para>
215         /// <para></para>
216         /// </returns>
217         [MethodImpl(MethodImplOptions.AggressiveInlining)]
218         protected override ref LinksHeader<TLink> GetHeaderReference() => ref
            ↪ AsRef<LinksHeader<TLink>>(_header);
219 
220         /// <summary>
221         /// <para>
222         /// Gets the link data part reference using the specified link index.
223         /// </para>
224         /// <para></para>
225         /// </summary>
226         /// <param name="linkIndex">
227         /// <para>The link index.</para>
228         /// <para></para>
229         /// </param>
230         /// <returns>
231         /// <para>A ref raw link data part of t link</para>
232         /// <para></para>
233         /// </returns>
234         [MethodImpl(MethodImplOptions.AggressiveInlining)]
235         protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
            ↪ => ref AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (LinkDataPartSizeInBytes *
            ↪ ConvertToInt64(linkIndex)));
236 
237         /// <summary>
238         /// <para>
239         /// Gets the link index part reference using the specified link index.
240         /// </para>
241         /// <para></para>
242         /// </summary>
243         /// <param name="linkIndex">
244         /// <para>The link index.</para>
245         /// <para></para>
246         /// </param>
247         /// <returns>
248         /// <para>A ref raw link index part of t link</para>
249         /// <para></para>
250         /// </returns>
251         [MethodImpl(MethodImplOptions.AggressiveInlining)]
252         protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
            ↪ linkIndex) => ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
            ↪ (LinkIndexPartSizeInBytes * ConvertToInt64(linkIndex)));
253     }
254 }
```

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;
```

```
using Platform.Converters;
using Platform.Numbers;
using Platform.Memory;
using Platform.Data.Exceptions;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the split memory links base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="DisposableBase"/>
    /// <seealso cref="ILinks{TLink}"/>
    public abstract class SplitMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;
        private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;
        private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
        ↪  UncheckedConverter<long, TLink>.Default;

        private static readonly TLink _zero = default;
        private static readonly TLink _one = Arithmetic.Increment(_zero);

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

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

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

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

        /// <summary>
        /// <para>
        /// The data memory.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly IResizableDirectMemory _dataMemory;
        /// <summary>
        /// <para>
        /// The index memory.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly IResizableDirectMemory _indexMemory;
        /// <summary>
        /// <para>
        /// The use linked list.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        protected readonly bool _useLinkedList;
        /// <summary>
        /// <para>
        /// The data memory reservation step in bytes.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly long _dataMemoryReservationStepInBytes;
        /// <summary>
        /// <para>
        /// The index memory reservation step in bytes.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly long _indexMemoryReservationStepInBytes;

        /// <summary>
        /// <para>
        /// The internal sources list methods.
        /// </para>
        /// <para></para>
        /// </summary>
        protected InternalLinksSourcesLinkedListMethods<TLink> InternalSourcesListMethods;
        /// <summary>
        /// <para>
        /// The internal sources tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        protected ILinksTreeMethods<TLink> InternalSourcesTreeMethods;
        /// <summary>
        /// <para>
        /// The external sources tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        protected ILinksTreeMethods<TLink> ExternalSourcesTreeMethods;
        /// <summary>
        /// <para>
        /// The internal targets tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        protected ILinksTreeMethods<TLink> InternalTargetsTreeMethods;
        /// <summary>
        /// <para>
        /// The external targets tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        protected ILinksTreeMethods<TLink> ExternalTargetsTreeMethods;
        // 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
        /// <summary>
        /// <para>
        /// The unused links list methods.
        /// </para>
        /// <para></para>
        /// </summary>
        protected ILinksListMethods<TLink> UnusedLinksListMethods;

        /// <summary>
        /// Возвращает общее число связей находящихся в хранилище.
        /// </summary>
        protected virtual TLink Total
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get
            {
                ref var header = ref GetHeaderReference();
                return Subtract(header.AllocatedLinks, header.FreeLinks);
            }
        }

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

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

```
231         /// <para></para>
232         /// </summary>
233         /// <param name="dataMemory">
234         /// <para>The data memory.</para>
235         /// <para></para>
236         /// </param>
237         /// <param name="indexMemory">
238         /// <para>The index memory.</para>
239         /// <para></para>
240         /// </param>
241         [MethodImpl(MethodImplOptions.AggressiveInlining)]
242         protected virtual void Init(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory)
243         {
244             // Read allocated links from header
245             if (indexMemory.ReservedCapacity < LinkHeaderSizeInBytes)
246             {
247                 indexMemory.ReservedCapacity = LinkHeaderSizeInBytes;
248             }
249             SetPointers(dataMemory, indexMemory);
250             ref var header = ref GetHeaderReference();
251             var allocatedLinks = ConvertToInt64(header.AllocatedLinks);
252             // Adjust reserved capacity
253             var minimumDataReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
254             if (minimumDataReservedCapacity < dataMemory.UsedCapacity)
255             {
256                 minimumDataReservedCapacity = dataMemory.UsedCapacity;
257             }
258             if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
259             {
260                 minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
261             }
262             var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
263             if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
264             {
265                 minimumIndexReservedCapacity = indexMemory.UsedCapacity;
266             }
267             if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
268             {
269                 minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
270             }
271             // Check for alignment
272             if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
273             {
274                 minimumDataReservedCapacity = ((minimumDataReservedCapacity /
                    ↪ _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
                    ↪ _dataMemoryReservationStepInBytes;
275             }
276             if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)
277             {
278                 minimumIndexReservedCapacity = ((minimumIndexReservedCapacity /
                    ↪ _indexMemoryReservationStepInBytes) * _indexMemoryReservationStepInBytes) +
                    ↪ _indexMemoryReservationStepInBytes;
279             }
280             if (dataMemory.ReservedCapacity != minimumDataReservedCapacity)
281             {
282                 dataMemory.ReservedCapacity = minimumDataReservedCapacity;
283             }
284             if (indexMemory.ReservedCapacity != minimumIndexReservedCapacity)
285             {
286                 indexMemory.ReservedCapacity = minimumIndexReservedCapacity;
287             }
288             SetPointers(dataMemory, indexMemory);
289             header = ref GetHeaderReference();
290             // Ensure correctness _memory.UsedCapacity over _header->AllocatedLinks
291             // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
292             dataMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
                ↪ LinkDataPartSizeInBytes) + LinkDataPartSizeInBytes; // First link is read only
                ↪ zero link.
293             indexMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
                ↪ LinkIndexPartSizeInBytes) + LinkHeaderSizeInBytes;
294             // Ensure correctness _memory.ReservedLinks over _header->ReservedCapacity
295             // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
296             header.ReservedLinks = ConvertToAddress((dataMemory.ReservedCapacity -
                ↪ LinkDataPartSizeInBytes) / LinkDataPartSizeInBytes);
297         }
298
299         /// <summary>
```

```
300         /// <para>
301         /// Counts the restrictions.
302         /// </para>
303         /// <para></para>
304         /// </summary>
305         /// <param name="restrictions">
306         /// <para>The restrictions.</para>
307         /// <para></para>
308         /// </param>
309         /// <exception cref="NotSupportedException">
310         /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
311         /// <para></para>
312         /// </exception>
313         /// <returns>
314         /// <para>The link</para>
315         /// <para></para>
316         /// </returns>
317         [MethodImpl(MethodImplOptions.AggressiveInlining)]
318         public virtual TLink Count(IList<TLink> restrictions)
319         {
320             // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
321             if (restrictions.Count == 0)
322             {
323                 return Total;
324             }
325             var constants = Constants;
326             var any = constants.Any;
327             var index = restrictions[constants.IndexPart];
328             if (restrictions.Count == 1)
329             {
330                 if (AreEqual(index, any))
331                 {
332                     return Total;
333                 }
334                 return Exists(index) ? GetOne() : GetZero();
335             }
336             if (restrictions.Count == 2)
337             {
338                 var value = restrictions[1];
339                 if (AreEqual(index, any))
340                 {
341                     if (AreEqual(value, any))
342                     {
343                         return Total; // Any - как отсутствие ограничения
344                     }
345                     var externalReferencesRange = constants.ExternalReferencesRange;
346                     if (externalReferencesRange.HasValue &&
                        ↪  externalReferencesRange.Value.Contains(value))
347                     {
348                         return Add(ExternalSourcesTreeMethods.CountUsages(value),
                            ↪  ExternalTargetsTreeMethods.CountUsages(value));
349                     }
350                     else
351                     {
352                         if (_useLinkedList)
353                         {
354                             return Add(InternalSourcesListMethods.CountUsages(value),
                                ↪  InternalTargetsTreeMethods.CountUsages(value));
355                         }
356                         else
357                         {
358                             return Add(InternalSourcesTreeMethods.CountUsages(value),
                                ↪  InternalTargetsTreeMethods.CountUsages(value));
359                         }
360                     }
361                 }
362                 else
363                 {
364                     if (!Exists(index))
365                     {
366                         return GetZero();
367                     }
368                     if (AreEqual(value, any))
369                     {
370                         return GetOne();
371                     }
372                     ref var storedLinkValue = ref GetLinkDataPartReference(index);
```

```
                    if (AreEqual(storedLinkValue.Source, value) ||
                    ↪  AreEqual(storedLinkValue.Target, value))
                    {
                        return GetOne();
                    }
                    return GetZero();
                }
            }
            if (restrictions.Count == 3)
            {
                var externalReferencesRange = constants.ExternalReferencesRange;
                var source = restrictions[constants.SourcePart];
                var target = restrictions[constants.TargetPart];
                if (AreEqual(index, any))
                {
                    if (AreEqual(source, any) && AreEqual(target, any))
                    {
                        return Total;
                    }
                    else if (AreEqual(source, any))
                    {
                        if (externalReferencesRange.HasValue &&
                        ↪  externalReferencesRange.Value.Contains(target))
                        {
                            return ExternalTargetsTreeMethods.CountUsages(target);
                        }
                        else
                        {
                            return InternalTargetsTreeMethods.CountUsages(target);
                        }
                    }
                    else if (AreEqual(target, any))
                    {
                        if (externalReferencesRange.HasValue &&
                        ↪  externalReferencesRange.Value.Contains(source))
                        {
                            return ExternalSourcesTreeMethods.CountUsages(source);
                        }
                        else
                        {
                            if (_useLinkedList)
                            {
                                return InternalSourcesListMethods.CountUsages(source);
                            }
                            else
                            {
                                return InternalSourcesTreeMethods.CountUsages(source);
                            }
                        }
                    }
                    else //if(source != Any && target != Any)
                    {
                        // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
                        TLink link;
                        if (externalReferencesRange.HasValue)
                        {
                            if (externalReferencesRange.Value.Contains(source) &&
                            ↪  externalReferencesRange.Value.Contains(target))
                            {
                                link = ExternalSourcesTreeMethods.Search(source, target);
                            }
                            else if (externalReferencesRange.Value.Contains(source))
                            {
                                link = InternalTargetsTreeMethods.Search(source, target);
                            }
                            else if (externalReferencesRange.Value.Contains(target))
                            {
                                if (_useLinkedList)
                                {
                                    link = ExternalSourcesTreeMethods.Search(source, target);
                                }
                                else
                                {
                                    link = InternalSourcesTreeMethods.Search(source, target);
                                }
                            }
                            else
                            {
```

```
                                if (_useLinkedList ||
                                    GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
                                    InternalTargetsTreeMethods.CountUsages(target)))
                                {
                                    link = InternalTargetsTreeMethods.Search(source, target);
                                }
                                else
                                {
                                    link = InternalSourcesTreeMethods.Search(source, target);
                                }
                            }
                        }
                        else
                        {
                            if (_useLinkedList ||
                                GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
                                InternalTargetsTreeMethods.CountUsages(target)))
                            {
                                link = InternalTargetsTreeMethods.Search(source, target);
                            }
                            else
                            {
                                link = InternalSourcesTreeMethods.Search(source, target);
                            }
                        }
                        return AreEqual(link, constants.Null) ? GetZero() : GetOne();
                    }
                }
                else
                {
                    if (!Exists(index))
                    {
                        return GetZero();
                    }
                    if (AreEqual(source, any) && AreEqual(target, any))
                    {
                        return GetOne();
                    }
                    ref var storedLinkValue = ref GetLinkDataPartReference(index);
                    if (!AreEqual(source, any) && !AreEqual(target, any))
                    {
                        if (AreEqual(storedLinkValue.Source, source) &&
                            AreEqual(storedLinkValue.Target, target))
                        {
                            return GetOne();
                        }
                        return GetZero();
                    }
                    var value = default(TLink);
                    if (AreEqual(source, any))
                    {
                        value = target;
                    }
                    if (AreEqual(target, any))
                    {
                        value = source;
                    }
                    if (AreEqual(storedLinkValue.Source, value) ||
                        AreEqual(storedLinkValue.Target, value))
                    {
                        return GetOne();
                    }
                    return GetZero();
                }
            }
            throw new NotSupportedException("Другие размеры и способы ограничений не
                поддерживаются.");
        }

        /// <summary>
        /// <para>
        /// Eaches the handler.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
```

```
518             /// </param>
519             /// <param name="restrictions">
520             /// <para>The restrictions.</para>
521             /// <para></para>
522             /// </param>
523             /// <exception cref="NotSupportedException">
524             /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
525             /// <para></para>
526             /// </exception>
527             /// <returns>
528             /// <para>The link</para>
529             /// <para></para>
530             /// </returns>
531             [MethodImpl(MethodImplOptions.AggressiveInlining)]
532             public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
533             {
534                 var constants = Constants;
535                 var @break = constants.Break;
536                 if (restrictions.Count == 0)
537                 {
538                     for (var link = GetOne(); LessOrEqualThan(link,
                        ↪  GetHeaderReference().AllocatedLinks); link = Increment(link))
539                     {
540                         if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
541                         {
542                             return @break;
543                         }
544                     }
545                     return @break;
546                 }
547                 var @continue = constants.Continue;
548                 var any = constants.Any;
549                 var index = restrictions[constants.IndexPart];
550                 if (restrictions.Count == 1)
551                 {
552                     if (AreEqual(index, any))
553                     {
554                         return Each(handler, Array.Empty<TLink>());
555                     }
556                     if (!Exists(index))
557                     {
558                         return @continue;
559                     }
560                     return handler(GetLinkStruct(index));
561                 }
562                 if (restrictions.Count == 2)
563                 {
564                     var value = restrictions[1];
565                     if (AreEqual(index, any))
566                     {
567                         if (AreEqual(value, any))
568                         {
569                             return Each(handler, Array.Empty<TLink>());
570                         }
571                         if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
572                         {
573                             return @break;
574                         }
575                         return Each(handler, new Link<TLink>(index, any, value));
576                     }
577                     else
578                     {
579                         if (!Exists(index))
580                         {
581                             return @continue;
582                         }
583                         if (AreEqual(value, any))
584                         {
585                             return handler(GetLinkStruct(index));
586                         }
587                         ref var storedLinkValue = ref GetLinkDataPartReference(index);
588                         if (AreEqual(storedLinkValue.Source, value) ||
589                             AreEqual(storedLinkValue.Target, value))
590                         {
591                             return handler(GetLinkStruct(index));
592                         }
593                         return @continue;
594                     }
```

```
595                 }
596                 if (restrictions.Count == 3)
597                 {
598                     var externalReferencesRange = constants.ExternalReferencesRange;
599                     var source = restrictions[constants.SourcePart];
600                     var target = restrictions[constants.TargetPart];
601                     if (AreEqual(index, any))
602                     {
603                         if (AreEqual(source, any) && AreEqual(target, any))
604                         {
605                             return Each(handler, Array.Empty<TLink>());
606                         }
607                         else if (AreEqual(source, any))
608                         {
609                             if (externalReferencesRange.HasValue &&
                                ↪  externalReferencesRange.Value.Contains(target))
610                             {
611                                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
612                             }
613                             else
614                             {
615                                 return InternalTargetsTreeMethods.EachUsage(target, handler);
616                             }
617                         }
618                         else if (AreEqual(target, any))
619                         {
620                             if (externalReferencesRange.HasValue &&
                                ↪  externalReferencesRange.Value.Contains(source))
621                             {
622                                 return ExternalSourcesTreeMethods.EachUsage(source, handler);
623                             }
624                             else
625                             {
626                                 if (_useLinkedList)
627                                 {
628                                     return InternalSourcesListMethods.EachUsage(source, handler);
629                                 }
630                                 else
631                                 {
632                                     return InternalSourcesTreeMethods.EachUsage(source, handler);
633                                 }
634                             }
635                         }
636                         else //if(source != Any && target != Any)
637                         {
638                             TLink link;
639                             if (externalReferencesRange.HasValue)
640                             {
641                                 if (externalReferencesRange.Value.Contains(source) &&
                                    ↪  externalReferencesRange.Value.Contains(target))
642                                 {
643                                     link = ExternalSourcesTreeMethods.Search(source, target);
644                                 }
645                                 else if (externalReferencesRange.Value.Contains(source))
646                                 {
647                                     link = InternalTargetsTreeMethods.Search(source, target);
648                                 }
649                                 else if (externalReferencesRange.Value.Contains(target))
650                                 {
651                                     if (_useLinkedList)
652                                     {
653                                         link = ExternalSourcesTreeMethods.Search(source, target);
654                                     }
655                                     else
656                                     {
657                                         link = InternalSourcesTreeMethods.Search(source, target);
658                                     }
659                                 }
660                                 else
661                                 {
662                                     if (_useLinkedList ||
                                        ↪  GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
                                        ↪  InternalTargetsTreeMethods.CountUsages(target)))
663                                     {
664                                         link = InternalTargetsTreeMethods.Search(source, target);
665                                     }
666                                     else
667                                     {
```

```
                                    link = InternalSourcesTreeMethods.Search(source, target);
                                }
                            }
                        }
                        else
                        {
                            if (_useLinkedList ||
                                GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
                                InternalTargetsTreeMethods.CountUsages(target)))
                            {
                                link = InternalTargetsTreeMethods.Search(source, target);
                            }
                            else
                            {
                                link = InternalSourcesTreeMethods.Search(source, target);
                            }
                        }
                        return AreEqual(link, constants.Null) ? @continue :
                            handler(GetLinkStruct(link));
                    }
                }
                else
                {
                    if (!Exists(index))
                    {
                        return @continue;
                    }
                    if (AreEqual(source, any) && AreEqual(target, any))
                    {
                        return handler(GetLinkStruct(index));
                    }
                    ref var storedLinkValue = ref GetLinkDataPartReference(index);
                    if (!AreEqual(source, any) && !AreEqual(target, any))
                    {
                        if (AreEqual(storedLinkValue.Source, source) &&
                            AreEqual(storedLinkValue.Target, target))
                        {
                            return handler(GetLinkStruct(index));
                        }
                        return @continue;
                    }
                    var value = default(TLink);
                    if (AreEqual(source, any))
                    {
                        value = target;
                    }
                    if (AreEqual(target, any))
                    {
                        value = source;
                    }
                    if (AreEqual(storedLinkValue.Source, value) ||
                        AreEqual(storedLinkValue.Target, value))
                    {
                        return handler(GetLinkStruct(index));
                    }
                    return @continue;
                }
            }
            throw new NotSupportedException("Другие размеры и способы ограничений не
                поддерживаются.");
        }

        /// <remarks>
        /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
            в другом месте (но не в менеджере памяти, а в логике Links)
        /// </remarks>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
        {
            var constants = Constants;
            var @null = constants.Null;
            var externalReferencesRange = constants.ExternalReferencesRange;
            var linkIndex = restrictions[constants.IndexPart];
            ref var link = ref GetLinkDataPartReference(linkIndex);
            var source = link.Source;
            var target = link.Target;
            ref var header = ref GetHeaderReference();
            ref var rootAsSource = ref header.RootAsSource;
```

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

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

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

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

```
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink GetZero() => default;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual bool AreEqual(TLink first, TLink second) =>
        ↪  _equalityComparer.Equals(first, second);

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
        ↪  second) < 0;

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
        ↪  _comparer.Compare(first, second) <= 0;

```

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

```
        protected virtual TLink ConvertToAddress(long value) =>
        ↪ _int64ToAddressConverter.Convert(value);

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
        ↪ second);

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink Subtract(TLink first, TLink second) =>
        ↪ Arithmetic<TLink>.Subtract(first, second);

        /// <summary>
        /// <para>
        /// Increments the link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);

        /// <summary>
        /// <para>
        /// Decrements the link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
```

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

1.44 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/UnusedLinksListMethods.cs

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

```
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UnusedLinksListMethods(byte* links, byte* header)
        {
            _links = links;
            _header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪  AsRef<LinksHeader<TLink>>(_header);

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
        ↪  AsRef<RawLinkDataPart<TLink>>(_links + (RawLinkDataPart<TLink>.SizeInBytes *
        ↪  _addressToInt64Converter.Convert(link)));

        /// <summary>
        /// <para>
        /// Gets the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;

        /// <summary>
        /// <para>
        /// Gets the last.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetLast() => GetHeaderReference().LastFreeLink;

        /// <summary>
        /// <para>
        /// Gets the previous using the specified element.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetPrevious(TLink element) =>
        ↪ GetLinkDataPartReference(element).Source;

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

        /// <summary>
        /// <para>
        /// Gets the size.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetSize() => GetHeaderReference().FreeLinks;

        /// <summary>
        /// <para>
        /// Sets the first using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
        ↪ element;

        /// <summary>
        /// <para>
        /// Sets the last using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
        ↪ element;

        /// <summary>
        /// <para>
        /// Sets the previous using the specified element.
        /// </para>
        /// <para></para>
```

```
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        /// <param name="previous">
        /// <para>The previous.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetPrevious(TLink element, TLink previous) =>
        ↪ GetLinkDataPartReference(element).Source = previous;

        /// <summary>
        /// <para>
        /// Sets the next using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        /// <param name="next">
        /// <para>The next.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetNext(TLink element, TLink next) =>
        ↪ GetLinkDataPartReference(element).Target = next;

        /// <summary>
        /// <para>
        /// Sets the size using the specified size.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
    }
}
```

1.45 ./csharp/Platform.Data.Doublets/Memory/Split/RawLinkDataPart.cs

```
using Platform.Unsafe;
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.Split
{
    /// <summary>
    /// <para>
    /// The raw link data part.
    /// </para>
    /// <para></para>
    /// </summary>
    public struct RawLinkDataPart<TLink> : IEquatable<RawLinkDataPart<TLink>>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪ EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// The size.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly long SizeInBytes = Structure<RawLinkDataPart<TLink>>.Size;

        /// <summary>
        /// <para>
        /// The source.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        public TLink Source;
        /// <summary>
        /// <para>
        /// The target.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink Target;

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="obj">
        /// <para>The obj.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override bool Equals(object obj) => obj is RawLinkDataPart<TLink> link ?
        ↪  Equals(link) : false;

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
        /// <para>The other.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Equals(RawLinkDataPart<TLink> other)
            => _equalityComparer.Equals(Source, other.Source)
            && _equalityComparer.Equals(Target, other.Target);

        /// <summary>
        /// <para>
        /// Gets the hash code.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override int GetHashCode() => (Source, Target).GetHashCode();

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

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool operator !=(RawLinkDataPart<TLink> left, RawLinkDataPart<TLink>
        ↪  right) => !(left == right);
    }
}
```

1.46 ./csharp/Platform.Data.Doublets/Memory/Split/RawLinkIndexPart.cs

```
using Platform.Unsafe;
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

```
7
8  namespace Platform.Data.Doublets.Memory.Split
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link index part.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLinkIndexPart<TLink> : IEquatable<RawLinkIndexPart<TLink>>
17     {
18         private static readonly EqualityComparer<TLink> _equalityComparer =
           ↪  EqualityComparer<TLink>.Default;
19
20         /// <summary>
21         /// <para>
22         /// The size.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         public static readonly long SizeInBytes = Structure<RawLinkIndexPart<TLink>>.Size;
27
28         /// <summary>
29         /// <para>
30         /// The root as source.
31         /// </para>
32         /// <para></para>
33         /// </summary>
34         public TLink RootAsSource;
35         /// <summary>
36         /// <para>
37         /// The left as source.
38         /// </para>
39         /// <para></para>
40         /// </summary>
41         public TLink LeftAsSource;
42         /// <summary>
43         /// <para>
44         /// The right as source.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         public TLink RightAsSource;
49         /// <summary>
50         /// <para>
51         /// The size as source.
52         /// </para>
53         /// <para></para>
54         /// </summary>
55         public TLink SizeAsSource;
56         /// <summary>
57         /// <para>
58         /// The root as target.
59         /// </para>
60         /// <para></para>
61         /// </summary>
62         public TLink RootAsTarget;
63         /// <summary>
64         /// <para>
65         /// The left as target.
66         /// </para>
67         /// <para></para>
68         /// </summary>
69         public TLink LeftAsTarget;
70         /// <summary>
71         /// <para>
72         /// The right as target.
73         /// </para>
74         /// <para></para>
75         /// </summary>
76         public TLink RightAsTarget;
77         /// <summary>
78         /// <para>
79         /// The size as target.
80         /// </para>
81         /// <para></para>
82         /// </summary>
83         public TLink SizeAsTarget;
84
```

```
        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="obj">
        /// <para>The obj.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override bool Equals(object obj) => obj is RawLinkIndexPart<TLink> link ?
        ↪  Equals(link) : false;

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
        /// <para>The other.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Equals(RawLinkIndexPart<TLink> other)
            => _equalityComparer.Equals(RootAsSource, other.RootAsSource)
            && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
            && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
            && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
            && _equalityComparer.Equals(RootAsTarget, other.RootAsTarget)
            && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
            && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
            && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);

        /// <summary>
        /// <para>
        /// Gets the hash code.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override int GetHashCode() => (RootAsSource, LeftAsSource, RightAsSource,
        ↪  SizeAsSource, RootAsTarget, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();

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

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool operator !=(RawLinkIndexPart<TLink> left, RawLinkIndexPart<TLink>
        ↪  right) => !(left == right);
    }
}
```

1.47 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksRecursionlessSizeBalancedTree

```
using System.Runtime.CompilerServices;
using Platform.Data.Doublets.Memory.Split.Generic;
using TLink = System.UInt32;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
```

```
    /// Represents the int 32 external links recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
    ↪ ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
    {
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly LinksHeader<TLink>* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪ cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected
        ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
        {
            LinksDataParts = linksDataParts;
            LinksIndexParts = linksIndexParts;
            Header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetZero() => 0U;
```

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

```
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThanZero(TLink value) => true; // value >= 0 is
        ↪  always true for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThanZero(TLink value) => value == 0UL; // value is
        ↪  always >= 0 for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;

        /// <summary>
```

```
        /// <para>
        /// Determines whether this instance less than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
        ↪  for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(TLink first, TLink second) => first < second;

        /// <summary>
        /// <para>
        /// Increments the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Increment(TLink value) => ++value;

        /// <summary>
        /// <para>
        /// Decrements the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Decrement(TLink value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
```

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

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

1.48 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSizeBalancedTreeMethodsBase

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

```
15      /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
16      /// <seealso cref="ILinksTreeMethods{TLink}"/>
17      public unsafe abstract class UInt32ExternalLinksSizeBalancedTreeMethodsBase :
        ↪  ExternalLinksSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
18      {
19          /// <summary>
20          /// <para>
21          /// The links data parts.
22          /// </para>
23          /// <para></para>
24          /// </summary>
25          protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
26          /// <summary>
27          /// <para>
28          /// The links index parts.
29          /// </para>
30          /// <para></para>
31          /// </summary>
32          protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
33          /// <summary>
34          /// <para>
35          /// The header.
36          /// </para>
37          /// <para></para>
38          /// </summary>
39          protected new readonly LinksHeader<TLink>* Header;
40
41          /// <summary>
42          /// <para>
43          /// Initializes a new <see cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
            ↪  instance.
44          /// </para>
45          /// <para></para>
46          /// </summary>
47          /// <param name="constants">
48          /// <para>A constants.</para>
49          /// <para></para>
50          /// </param>
51          /// <param name="linksDataParts">
52          /// <para>A links data parts.</para>
53          /// <para></para>
54          /// </param>
55          /// <param name="linksIndexParts">
56          /// <para>A links index parts.</para>
57          /// <para></para>
58          /// </param>
59          /// <param name="header">
60          /// <para>A header.</para>
61          /// <para></para>
62          /// </param>
63          [MethodImpl(MethodImplOptions.AggressiveInlining)]
64          protected UInt32ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
            ↪  constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
            ↪  linksIndexParts, LinksHeader<TLink>* header)
65              : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
66          {
67              LinksDataParts = linksDataParts;
68              LinksIndexParts = linksIndexParts;
69              Header = header;
70          }
71
72          /// <summary>
73          /// <para>
74          /// Gets the zero.
75          /// </para>
76          /// <para></para>
77          /// </summary>
78          /// <returns>
79          /// <para>The link</para>
80          /// <para></para>
81          /// </returns>
82          [MethodImpl(MethodImplOptions.AggressiveInlining)]
83          protected override TLink GetZero() => OU;
84
85          /// <summary>
86          /// <para>
87          /// Determines whether this instance equal to zero.
88          /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(TLink value) => value == OU;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(TLink first, TLink second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(TLink value) => value > OU;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(TLink first, TLink second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThanZero(TLink value) => true; // value >= 0 is
        ↪  always true for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThanZero(TLink value) => value == 0UL; // value is
        ↪  always >= 0 for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than zero.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThanZero(TLink value) => false; // value < 0 is always false
        ↪  for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(TLink first, TLink second) => first < second;

        /// <summary>
        /// <para>
        /// Increments the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Increment(TLink value) => ++value;

        /// <summary>
        /// <para>
        /// Decrements the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Decrement(TLink value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
```

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

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

1.49 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesRecursionlessSizeBalanc

```
1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links sources recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
       ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
```

```
        /// <para>
        /// Initializes a new <see
        ↪  cref="UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public
        ↪  UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref TLink GetLeftReference(TLink node) => ref
        ↪  LinksIndexParts[node].LeftAsSource;

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

        /// <summary>
        /// <para>
        /// Gets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <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) =>
            ↪ LinksIndexParts[node].LeftAsSource = left;
126
127         /// <summary>
128         /// <para>
129         /// Sets the right using the specified node.
130         /// </para>
131         /// <para></para>
132         /// </summary>
133         /// <param name="node">
134         /// <para>The node.</para>
135         /// <para></para>
136         /// </param>
137         /// <param name="right">
138         /// <para>The right.</para>
139         /// <para></para>
140         /// </param>
141         [MethodImpl(MethodImplOptions.AggressiveInlining)]
142         protected override void SetRight(TLink node, TLink right) =>
            ↪ LinksIndexParts[node].RightAsSource = right;
143
144         /// <summary>
145         /// <para>
146         /// Gets the size using the specified node.
147         /// </para>
148         /// <para></para>
149         /// </summary>
150         /// <param name="node">
151         /// <para>The node.</para>
152         /// <para></para>
153         /// </param>
154         /// <returns>
155         /// <para>The link</para>
156         /// <para></para>
157         /// </returns>
158         [MethodImpl(MethodImplOptions.AggressiveInlining)]
159         protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
160
161         /// <summary>
162         /// <para>
163         /// Sets the size using the specified node.
164         /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  LinksIndexParts[node].SizeAsSource = size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetTreeRoot() => Header->RootAsSource;

        /// <summary>
        /// <para>
        /// Gets the base part value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget)
            => firstSource < secondSource || firstSource == secondSource && firstTarget <
            ↪  secondTarget;

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

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

1.50 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksSourcesSizeBalancedTreeMeth

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

```
25          /// <para></para>
26          /// </param>
27          /// <param name="linksDataParts">
28          /// <para>A links data parts.</para>
29          /// <para></para>
30          /// </param>
31          /// <param name="linksIndexParts">
32          /// <para>A links index parts.</para>
33          /// <para></para>
34          /// </param>
35          /// <param name="header">
36          /// <para>A header.</para>
37          /// <para></para>
38          /// </param>
39          [MethodImpl(MethodImplOptions.AggressiveInlining)]
40          public UInt32ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<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>
```

```
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsSource;

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  LinksIndexParts[node].LeftAsSource = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  LinksIndexParts[node].RightAsSource = right;

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

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
```

```
174             /// </param>
175             [MethodImpl(MethodImplOptions.AggressiveInlining)]
176             protected override void SetSize(TLink node, TLink size) =>
                ↪  LinksIndexParts[node].SizeAsSource = size;
177
178             /// <summary>
179             /// <para>
180             /// Gets the tree root.
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 FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
                ↪  TLink secondSource, TLink secondTarget)
236                 => firstSource < secondSource || firstSource == secondSource && firstTarget <
                    ↪  secondTarget;
237
238             /// <summary>
239             /// <para>
240             /// Determines whether this instance first is to the right of second.
241             /// </para>
242             /// <para></para>
243             /// </summary>
244             /// <param name="firstSource">
245             /// <para>The first source.</para>
246             /// <para></para>
247             /// </param>
248             /// <param name="firstTarget">
```

```
        /// <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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget)
            => firstSource > secondSource || firstSource == secondSource && firstTarget >
            ↪  secondTarget;

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

1.51 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsRecursionlessSizeBalanc

```
using System.Runtime.CompilerServices;
using TLink = System.UInt32;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 external links targets recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↪  UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
```

```
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;

        /// <summary>
        /// <para>
        /// Gets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
```

```
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  LinksIndexParts[node].LeftAsTarget = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  LinksIndexParts[node].RightAsTarget = right;

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

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  LinksIndexParts[node].SizeAsTarget = size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
```

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

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

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

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

```
        /// <para>The second target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget)
            => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
            ↪  secondSource;

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

1.52 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32ExternalLinksTargetsSizeBalancedTreeMetho

```
using System.Runtime.CompilerServices;
using TLink = System.UInt32;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 external links targets size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32ExternalLinksTargetsSizeBalancedTreeMethods :
    ↪  UInt32ExternalLinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32ExternalLinksTargetsSizeBalancedTreeMethods"/>
        ↪  instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
40          public UInt32ExternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
            ↪  constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
            ↪  linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
            ↪  linksIndexParts, header) { }
41
42          /// <summary>
43          /// <para>
44          /// Gets the left reference using the specified node.
45          /// </para>
46          /// <para></para>
47          /// </summary>
48          /// <param name="node">
49          /// <para>The node.</para>
50          /// <para></para>
51          /// </param>
52          /// <returns>
53          /// <para>The ref link</para>
54          /// <para></para>
55          /// </returns>
56          [MethodImpl(MethodImplOptions.AggressiveInlining)]
57          protected override ref TLink GetLeftReference(TLink node) => ref
            ↪  LinksIndexParts[node].LeftAsTarget;
58
59          /// <summary>
60          /// <para>
61          /// Gets the right reference using the specified node.
62          /// </para>
63          /// <para></para>
64          /// </summary>
65          /// <param name="node">
66          /// <para>The node.</para>
67          /// <para></para>
68          /// </param>
69          /// <returns>
70          /// <para>The ref link</para>
71          /// <para></para>
72          /// </returns>
73          [MethodImpl(MethodImplOptions.AggressiveInlining)]
74          protected override ref TLink GetRightReference(TLink node) => ref
            ↪  LinksIndexParts[node].RightAsTarget;
75
76          /// <summary>
77          /// <para>
78          /// Gets the left using the specified node.
79          /// </para>
80          /// <para></para>
81          /// </summary>
82          /// <param name="node">
83          /// <para>The node.</para>
84          /// <para></para>
85          /// </param>
86          /// <returns>
87          /// <para>The link</para>
88          /// <para></para>
89          /// </returns>
90          [MethodImpl(MethodImplOptions.AggressiveInlining)]
91          protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93          /// <summary>
94          /// <para>
95          /// Gets the right using the specified node.
96          /// </para>
97          /// <para></para>
98          /// </summary>
99          /// <param name="node">
100         /// <para>The node.</para>
101         /// <para></para>
102         /// </param>
103         /// <returns>
104         /// <para>The link</para>
105         /// <para></para>
106         /// </returns>
107         [MethodImpl(MethodImplOptions.AggressiveInlining)]
108         protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110         /// <summary>
111         /// <para>
112         /// Sets the left using the specified node.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪ LinksIndexParts[node].LeftAsTarget = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪ LinksIndexParts[node].RightAsTarget = right;

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

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪ LinksIndexParts[node].SizeAsTarget = size;

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

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

1.53 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/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 :
       ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
17     {
18         /// <summary>
19         /// <para>
20         /// The links data parts.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
25         /// <summary>
26         /// <para>
27         /// The links index parts.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
32         /// <summary>
33         /// <para>
34         /// The header.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         protected new readonly LinksHeader<TLink>* Header;
39 
40         /// <summary>
41         /// <para>
42         /// Initializes a new <see
           ↪ cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="constants">
47         /// <para>A constants.</para>
48         /// <para></para>
49         /// </param>
```

```
50          /// <param name="linksDataParts">
51          /// <para>A links data parts.</para>
52          /// <para></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
            ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
            ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
            ↪ linksIndexParts, LinksHeader<TLink>* header)
64              : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
65          {
66              LinksDataParts = linksDataParts;
67              LinksIndexParts = linksIndexParts;
68              Header = header;
69          }
70
71          /// <summary>
72          /// <para>
73          /// Gets the zero.
74          /// </para>
75          /// <para></para>
76          /// </summary>
77          /// <returns>
78          /// <para>The 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 == 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>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Increment(TLink value) => ++value;

        /// <summary>
        /// <para>
        /// Decrements the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Decrement(TLink value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Add(TLink first, TLink second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Subtract(TLink first, TLink second) => first - second;

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪  ref LinksDataParts[link];

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪  ref LinksIndexParts[link];

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

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

```
using System.Runtime.CompilerServices;
using Platform.Data.Doublets.Memory.Split.Generic;
using TLink = System.UInt32;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 internal links size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe abstract class UInt32InternalLinksSizeBalancedTreeMethodsBase :
    ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly LinksHeader<TLink>* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
        ↪ instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected UInt32InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪ linksIndexParts, LinksHeader<TLink>* header)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
        {
            LinksDataParts = linksDataParts;
            LinksIndexParts = linksIndexParts;
            Header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetZero() => 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(TLink value) => value == 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(TLink first, TLink second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(TLink value) => value > 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </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) =>
            ↪  ref LinksDataParts[link];
366
367         /// <summary>
368         /// <para>
369         /// Gets the link index part reference using the specified link.
370         /// </para>
371         /// <para></para>
372         /// </summary>
373         /// <param name="link">
374         /// <para>The link.</para>
375         /// <para></para>
376         /// </param>
377         /// <returns>
378         /// <para>A ref raw link index part of t link</para>
379         /// <para></para>
380         /// </returns>
381         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

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

1.55 ./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 :
       ↪ InternalLinksSourcesLinkedListMethods<TLink>
16     {
17         private readonly RawLinkDataPart<TLink>* _linksDataParts;
18         private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
19
20         /// <summary>
21         /// <para>
22         /// Initializes a new <see cref="UInt32InternalLinksSourcesLinkedListMethods"/> instance.
23         /// </para>
24         /// <para></para>
25         /// </summary>
26         /// <param name="constants">
27         /// <para>A constants.</para>
```

```
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt32InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
        ↪  RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
        {
            _linksDataParts = linksDataParts;
            _linksIndexParts = linksIndexParts;
        }

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪  ref _linksDataParts[link];

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪  ref _linksIndexParts[link];
    }
}
```

1.56 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesRecursionlessSizeBalanc

```
using System.Runtime.CompilerServices;
using TLink = System.UInt32;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 internal links sources recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
    ↪  UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <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) =>
            ↪ LinksIndexParts[node].LeftAsSource = left;
126
127         /// <summary>
128         /// <para>
129         /// Sets the right using the specified node.
130         /// </para>
131         /// <para></para>
132         /// </summary>
133         /// <param name="node">
134         /// <para>The node.</para>
135         /// <para></para>
136         /// </param>
137         /// <param name="right">
138         /// <para>The right.</para>
139         /// <para></para>
140         /// </param>
141         [MethodImpl(MethodImplOptions.AggressiveInlining)]
142         protected override void SetRight(TLink node, TLink right) =>
            ↪ LinksIndexParts[node].RightAsSource = right;
143
144         /// <summary>
145         /// <para>
146         /// Gets the size using the specified node.
147         /// </para>
148         /// <para></para>
149         /// </summary>
150         /// <param name="node">
151         /// <para>The node.</para>
152         /// <para></para>
153         /// </param>
154         /// <returns>
155         /// <para>The link</para>
156         /// <para></para>
157         /// </returns>
158         [MethodImpl(MethodImplOptions.AggressiveInlining)]
159         protected override TLink GetSize(TLink node) => LinksIndexParts[node].SizeAsSource;
160
161         /// <summary>
162         /// <para>
163         /// Sets the size using the specified node.
164         /// </para>
165         /// <para></para>
```

```
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  LinksIndexParts[node].SizeAsSource = size;

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

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

        /// <summary>
        /// <para>
        /// Gets the key part value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;

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

```
            link.LeftAsSource = Zero;
            link.RightAsSource = Zero;
            link.SizeAsSource = Zero;
        }

        /// <summary>
        /// <para>
        /// Searches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public override TLink Search(TLink source, TLink target) =>
        ↪ SearchCore(GetTreeRoot(source), target);
    }
}
```

1.57 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksSourcesSizeBalancedTreeMetho

```
using System.Runtime.CompilerServices;
using TLink = System.UInt32;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 internal links sources size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32InternalLinksSourcesSizeBalancedTreeMethods :
    ↪ UInt32InternalLinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32InternalLinksSourcesSizeBalancedTreeMethods"/>
        ↪ instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt32InternalLinksSourcesSizeBalancedTreeMethods(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>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref TLink GetLeftReference(TLink node) => ref
        ↪ LinksIndexParts[node].LeftAsSource;

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

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

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
```

```
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  LinksIndexParts[node].LeftAsSource = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  LinksIndexParts[node].RightAsSource = right;

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

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  LinksIndexParts[node].SizeAsSource = size;

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

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

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

        /// <summary>
        /// <para>
        /// Gets the key part value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;

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

        /// <summary>
        /// <para>
        /// Searches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public override TLink Search(TLink source, TLink target) =>
        ↪  SearchCore(GetTreeRoot(source), target);
    }
}
```

1.58 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsRecursionlessSizeBalanc

```
using System.Runtime.CompilerServices;
using TLink = System.UInt32;

```

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 internal links targets recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public
        ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></para>
        /// </returns>
        [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) =>
            ↪ LinksIndexParts[node].RightAsTarget = right;
143
144         /// <summary>
145         /// <para>
146         /// Gets the size using the specified node.
147         /// </para>
148         /// <para></para>
```

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

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Source;

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(TLink node)
        {
            ref var link = ref LinksIndexParts[node];
            link.LeftAsTarget = Zero;
            link.RightAsTarget = Zero;
            link.SizeAsTarget = Zero;
        }

        /// <summary>
        /// <para>
        /// Searches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public override TLink Search(TLink source, TLink target) =>
        ↪  SearchCore(GetTreeRoot(target), source);
    }
}
```

1.59 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32InternalLinksTargetsSizeBalancedTreeMetho

```
using System.Runtime.CompilerServices;
using TLink = System.UInt32;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 internal links targets size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32InternalLinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32InternalLinksTargetsSizeBalancedTreeMethods :
    ↪  UInt32InternalLinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32InternalLinksTargetsSizeBalancedTreeMethods"/>
        ↪  instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
```

```
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt32InternalLinksTargetsSizeBalancedTreeMethods(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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;

        /// <summary>
        /// <para>
        /// Gets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

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

```
        /// <para>
        /// Gets the tree root using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsTarget;

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

        /// <summary>
        /// <para>
        /// Gets the key part value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Source;

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(TLink node)
        {
            ref var link = ref LinksIndexParts[node];
            link.LeftAsTarget = Zero;
            link.RightAsTarget = Zero;
            link.SizeAsTarget = Zero;
        }

        /// <summary>
        /// <para>
        /// Searches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
```

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

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

```
1  using System;
2  using System.Runtime.CompilerServices;
3  using Platform.Singletons;
4  using Platform.Memory;
5  using Platform.Data.Doublets.Memory.Split.Generic;
6  using TLink = System.UInt32;
7
8  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10 namespace Platform.Data.Doublets.Memory.Split.Specific
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the int 32 split memory links.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
19     public unsafe class UInt32SplitMemoryLinks : SplitMemoryLinksBase<TLink>
20     {
21         private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
22         private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
23         private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
24         private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
25         private LinksHeader<TLink>* _header;
26         private RawLinkDataPart<TLink>* _linksDataParts;
27         private RawLinkIndexPart<TLink>* _linksIndexParts;
28
29         /// <summary>
30         /// <para>
31         /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         /// <param name="dataMemory">
36         /// <para>A data memory.</para>
37         /// <para></para>
38         /// </param>
39         /// <param name="indexMemory">
40         /// <para>A index memory.</para>
41         /// <para></para>
42         /// </param>
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46         /// <summary>
47         /// <para>
48         /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <param name="dataMemory">
53         /// <para>A data memory.</para>
54         /// <para></para>
55         /// </param>
56         /// <param name="indexMemory">
57         /// <para>A index memory.</para>
58         /// <para></para>
59         /// </param>
60         /// <param name="memoryReservationStep">
61         /// <para>A memory reservation step.</para>
62         /// <para></para>
63         /// </param>
```

```
64          [MethodImpl(MethodImplOptions.AggressiveInlining)]
65          public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪  indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
            ↪  memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
            ↪  IndexTreeType.Default, useLinkedList: true) { }
66
67          /// <summary>
68          /// <para>
69          /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
70          /// </para>
71          /// <para></para>
72          /// </summary>
73          /// <param name="dataMemory">
74          /// <para>A data memory.</para>
75          /// <para></para>
76          /// </param>
77          /// <param name="indexMemory">
78          /// <para>A index memory.</para>
79          /// <para></para>
80          /// </param>
81          /// <param name="memoryReservationStep">
82          /// <para>A memory reservation step.</para>
83          /// <para></para>
84          /// </param>
85          /// <param name="constants">
86          /// <para>A constants.</para>
87          /// <para></para>
88          /// </param>
89          [MethodImpl(MethodImplOptions.AggressiveInlining)]
90          public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪  indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
            ↪  this(dataMemory, indexMemory, memoryReservationStep, constants,
            ↪  IndexTreeType.Default, useLinkedList: true) { }
91
92          /// <summary>
93          /// <para>
94          /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
95          /// </para>
96          /// <para></para>
97          /// </summary>
98          /// <param name="dataMemory">
99          /// <para>A data memory.</para>
100         /// <para></para>
101         /// </param>
102         /// <param name="indexMemory">
103         /// <para>A index memory.</para>
104         /// <para></para>
105         /// </param>
106         /// <param name="memoryReservationStep">
107         /// <para>A memory reservation step.</para>
108         /// <para></para>
109         /// </param>
110         /// <param name="constants">
111         /// <para>A constants.</para>
112         /// <para></para>
113         /// </param>
114         /// <param name="indexTreeType">
115         /// <para>A index tree type.</para>
116         /// <para></para>
117         /// </param>
118         /// <param name="useLinkedList">
119         /// <para>A use linked list.</para>
120         /// <para></para>
121         /// </param>
122         [MethodImpl(MethodImplOptions.AggressiveInlining)]
123         public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪  indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
            ↪  IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
            ↪  memoryReservationStep, constants, useLinkedList)
124         {
125             if (indexTreeType == IndexTreeType.SizeBalancedTree)
126             {
127                 _createInternalSourceTreeMethods = () => new
                    ↪  UInt32InternalLinksSourcesSizeBalancedTreeMethods(Constants,
                    ↪  _linksDataParts, _linksIndexParts, _header);
128                 _createExternalSourceTreeMethods = () => new
                    ↪  UInt32ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
                    ↪  _linksDataParts, _linksIndexParts, _header);
```

```
129                 _createInternalTargetTreeMethods = () => new
                    ↪ UInt32InternalLinksTargetsSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
130                 _createExternalTargetTreeMethods = () => new
                    ↪ UInt32ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
131             }
132             else
133             {
134                 _createInternalSourceTreeMethods = () => new
                    ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
135                 _createExternalSourceTreeMethods = () => new
                    ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
136                 _createInternalTargetTreeMethods = () => new
                    ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
137                 _createExternalTargetTreeMethods = () => new
                    ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
138             }
139             Init(dataMemory, indexMemory);
140         }
141
142         /// <summary>
143         /// <para>
144         /// Sets the pointers using the specified data memory.
145         /// </para>
146         /// <para></para>
147         /// </summary>
148         /// <param name="dataMemory">
149         /// <para>The data memory.</para>
150         /// <para></para>
151         /// </param>
152         /// <param name="indexMemory">
153         /// <para>The index memory.</para>
154         /// <para></para>
155         /// </param>
156         [MethodImpl(MethodImplOptions.AggressiveInlining)]
157         protected override void SetPointers(IResizableDirectMemory dataMemory,
            ↪ IResizableDirectMemory indexMemory)
158         {
159             _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
160             _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
161             _header = (LinksHeader<TLink>*)indexMemory.Pointer;
162             if (_useLinkedList)
163             {
164                 InternalSourcesListMethods = new
                    ↪ UInt32InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
                    ↪ _linksIndexParts);
165             }
166             else
167             {
168                 InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
169             }
170             ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
171             InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
172             ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
173             UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_linksDataParts, _header);
174         }
175
176         /// <summary>
177         /// <para>
178         /// Resets the pointers.
179         /// </para>
180         /// <para></para>
181         /// </summary>
182         [MethodImpl(MethodImplOptions.AggressiveInlining)]
183         protected override void ResetPointers()
184         {
185             base.ResetPointers();
186             _linksDataParts = null;
187             _linksIndexParts = null;
188             _header = null;
189         }
190
191         /// <summary>
```

```
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link index.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
        ↪  => ref _linksDataParts[linkIndex];

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link index.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
        ↪  linkIndex) => ref _linksIndexParts[linkIndex];

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(TLink first, TLink second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
```

```
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(TLink first, TLink second) => first < second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(TLink first, TLink second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(TLink first, TLink second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(TLink first, TLink second) => first >= second;

        /// <summary>
        /// <para>
        /// Gets the zero.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetZero() => 0U;

        /// <summary>
        /// <para>
        /// Gets the one.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetOne() => 1U;

        /// <summary>
        /// <para>
        /// Converts the to int 64 using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The long</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override long ConvertToInt64(TLink value) => value;

        /// <summary>
        /// <para>
        /// Converts the to address using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink ConvertToAddress(long value) => (TLink)value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Add(TLink first, TLink second) => first + second;

```

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

1.61 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt32UnusedLinksListMethods.cs

```
1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt32;
4 
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6 
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 32 unused links list methods.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="UnusedLinksListMethods{TLink}"/>
16     public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<TLink>
17     {
18         private readonly RawLinkDataPart<TLink>* _links;
19         private readonly LinksHeader<TLink>* _header;
20 
21         /// <summary>
```

```
        /// <para>
        /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt32UnusedLinksListMethods(RawLinkDataPart<TLink>* links, LinksHeader<TLink>*
        ↪  header)
            : base((byte*)links, (byte*)header)
        {
            _links = links;
            _header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪  ref _links[link];

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
    }
}
```

1.62 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksRecursionlessSizeBalancedTree

```
using System.Runtime.CompilerServices;
using Platform.Data.Doublets.Memory.Split.Generic;
using TLink = System.UInt64;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 external links recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase :
    ↪  ExternalLinksRecursionlessSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
    {
        /// <summary>
        /// <para>
        /// The links data parts.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly LinksHeader<TLink>* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected
        ↪  UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↪  constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪  linksIndexParts, LinksHeader<TLink>* header)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
        {
            LinksDataParts = linksDataParts;
            LinksIndexParts = linksIndexParts;
            Header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetZero() => 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

```
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(ulong value) => value == 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(ulong first, ulong second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(ulong value) => value > 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(ulong first, ulong second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
```

```
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
        ↪  always true for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
        ↪  always >= 0 for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <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>
```

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Add(ulong first, ulong second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Subtract(ulong first, ulong second) => first - second;

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪  ref LinksDataParts[link];

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪  ref LinksIndexParts[link];

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

1.63 ./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 :
        ↪  ExternalLinksSizeBalancedTreeMethodsBase<TLink>, ILinksTreeMethods<TLink>
18      {
19          /// <summary>
20          /// <para>
21          /// The links data parts.
22          /// </para>
23          /// <para></para>
24          /// </summary>
25          protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
```

```
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly LinksHeader<TLink>* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
        ↪  instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected UInt64ExternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↪  constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪  linksIndexParts, LinksHeader<TLink>* header)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
        {
            LinksDataParts = linksDataParts;
            LinksIndexParts = linksIndexParts;
            Header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetZero() => 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(ulong value) => value == 0UL;
```

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

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
        ↪  always true for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
        ↪  always >= 0 for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
        ↪  for ulong

```

```
        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(ulong first, ulong second) => first < second;

        /// <summary>
        /// <para>
        /// Increments the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Increment(ulong value) => ++value;

        /// <summary>
        /// <para>
        /// Decrements the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Decrement(ulong value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Add(ulong first, ulong second) => first + second;

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

```
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Subtract(ulong first, ulong second) => first - second;

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<TLink> GetHeaderReference() => ref *Header;

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪  ref LinksDataParts[link];

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪  ref LinksIndexParts[link];

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

1.64 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksSourcesRecursionlessSizeBalanc

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

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

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

```
177
178         /// <summary>
179         /// <para>
180         /// Gets the tree root.
181         /// </para>
182         /// <para></para>
183         /// </summary>
184         /// <returns>
185         /// <para>The link</para>
186         /// <para></para>
187         /// </returns>
188         [MethodImpl(MethodImplOptions.AggressiveInlining)]
189         protected override 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,
            ↪  TLink secondSource, TLink secondTarget)
236             => firstSource < secondSource || firstSource == secondSource && firstTarget <
                ↪  secondTarget;
237
238         /// <summary>
239         /// <para>
240         /// Determines whether this instance first is to the right of second.
241         /// </para>
242         /// <para></para>
243         /// </summary>
244         /// <param name="firstSource">
245         /// <para>The first source.</para>
246         /// <para></para>
247         /// </param>
248         /// <param name="firstTarget">
249         /// <para>The first target.</para>
250         /// <para></para>
251         /// </param>
252         /// <param name="secondSource">
```

```
253          /// <para>The second source.</para>
254          /// <para></para>
255          /// </param>
256          /// <param name="secondTarget">
257          /// <para>The second target.</para>
258          /// <para></para>
259          /// </param>
260          /// <returns>
261          /// <para>The bool</para>
262          /// <para></para>
263          /// </returns>
264          [MethodImpl(MethodImplOptions.AggressiveInlining)]
265          protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
             ↪  TLink secondSource, TLink secondTarget)
266              => firstSource > secondSource || firstSource == secondSource && firstTarget >
                 ↪  secondTarget;
267
268          /// <summary>
269          /// <para>
270          /// Clears the node using the specified node.
271          /// </para>
272          /// <para></para>
273          /// </summary>
274          /// <param name="node">
275          /// <para>The node.</para>
276          /// <para></para>
277          /// </param>
278          [MethodImpl(MethodImplOptions.AggressiveInlining)]
279          protected override void ClearNode(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/UInt64ExternalLinksSourcesSizeBalancedTreeMeth

```
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 :
        ↪  UInt64ExternalLinksSizeBalancedTreeMethodsBase
16      {
17          /// <summary>
18          /// <para>
19          /// Initializes a new <see cref="UInt64ExternalLinksSourcesSizeBalancedTreeMethods"/>
            ↪  instance.
20          /// </para>
21          /// <para></para>
22          /// </summary>
23          /// <param name="constants">
24          /// <para>A constants.</para>
25          /// <para></para>
26          /// </param>
27          /// <param name="linksDataParts">
28          /// <para>A links data parts.</para>
29          /// <para></para>
30          /// </param>
31          /// <param name="linksIndexParts">
32          /// <para>A links index parts.</para>
33          /// <para></para>
34          /// </param>
35          /// <param name="header">
36          /// <para>A header.</para>
37          /// <para></para>
```

```
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt64ExternalLinksSourcesSizeBalancedTreeMethods(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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref TLink GetLeftReference(TLink node) => ref
        ↪  LinksIndexParts[node].LeftAsSource;

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

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

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

        /// <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].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) =>
            ↪ LinksIndexParts[node].RightAsSource = right;
143
144         /// <summary>
145         /// <para>
146         /// Gets the size using the specified node.
147         /// </para>
148         /// <para></para>
149         /// </summary>
150         /// <param name="node">
151         /// <para>The node.</para>
152         /// <para></para>
153         /// </param>
154         /// <returns>
155         /// <para>The link</para>
156         /// <para></para>
157         /// </returns>
158         [MethodImpl(MethodImplOptions.AggressiveInlining)]
159         protected override TLink GetSize(TLink node) => 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) =>
            ↪ LinksIndexParts[node].SizeAsSource = size;
177
178         /// <summary>
179         /// <para>
180         /// Gets the tree root.
181         /// </para>
182         /// <para></para>
183         /// </summary>
184         /// <returns>
185         /// <para>The link</para>
```

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

        /// <summary>
        /// <para>
        /// Gets the base part value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink 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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget)
            => firstSource > secondSource || firstSource == secondSource && firstTarget >
            ↪  secondTarget;

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

1.66 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsRecursionlessSizeBalanc

```
using System.Runtime.CompilerServices;
using TLink = System.UInt64;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 external links targets recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↪  UInt64ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public
        ↪  UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
```

```
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  LinksIndexParts[node].LeftAsTarget = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  LinksIndexParts[node].RightAsTarget = right;

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

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  LinksIndexParts[node].SizeAsTarget = size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetTreeRoot() => Header->RootAsTarget;

        /// <summary>
        /// <para>
        /// Gets the base part value using the specified node.
        /// </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,
          ↪  TLink secondSource, TLink secondTarget)
236             => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
              ↪  secondSource;
237
238         /// <summary>
239         /// <para>
240         /// Determines whether this instance first is to the right of second.
241         /// </para>
242         /// <para></para>
243         /// </summary>
244         /// <param name="firstSource">
245         /// <para>The first source.</para>
246         /// <para></para>
247         /// </param>
248         /// <param name="firstTarget">
249         /// <para>The first target.</para>
250         /// <para></para>
251         /// </param>
252         /// <param name="secondSource">
253         /// <para>The second source.</para>
254         /// <para></para>
255         /// </param>
256         /// <param name="secondTarget">
257         /// <para>The second target.</para>
258         /// <para></para>
259         /// </param>
260         /// <returns>
261         /// <para>The bool</para>
262         /// <para></para>
263         /// </returns>
264         [MethodImpl(MethodImplOptions.AggressiveInlining)]
265         protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
          ↪  TLink secondSource, TLink secondTarget)
266             => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
              ↪  secondSource;
267
268         /// <summary>
```

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

1.67 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64ExternalLinksTargetsSizeBalancedTreeMetho

```
using System.Runtime.CompilerServices;
using TLink = System.UInt64;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 external links targets size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64ExternalLinksTargetsSizeBalancedTreeMethods :
    ↪  UInt64ExternalLinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64ExternalLinksTargetsSizeBalancedTreeMethods"/>
        ↪  instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt64ExternalLinksTargetsSizeBalancedTreeMethods(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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <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) =>
            ↪ LinksIndexParts[node].LeftAsTarget = left;
126
127         /// <summary>
128         /// <para>
```

```
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  LinksIndexParts[node].RightAsTarget = right;

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

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  LinksIndexParts[node].SizeAsTarget = size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetTreeRoot() => Header->RootAsTarget;

        /// <summary>
        /// <para>
        /// Gets the base part value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </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,
              ↪  TLink secondSource, TLink secondTarget)
236             => firstTarget < secondTarget || firstTarget == secondTarget && firstSource <
                  ↪  secondSource;
237
238         /// <summary>
239         /// <para>
240         /// Determines whether this instance first is to the right of second.
241         /// </para>
242         /// <para></para>
243         /// </summary>
244         /// <param name="firstSource">
245         /// <para>The first source.</para>
246         /// <para></para>
247         /// </param>
248         /// <param name="firstTarget">
249         /// <para>The first target.</para>
250         /// <para></para>
251         /// </param>
252         /// <param name="secondSource">
253         /// <para>The second source.</para>
254         /// <para></para>
255         /// </param>
256         /// <param name="secondTarget">
257         /// <para>The second target.</para>
258         /// <para></para>
259         /// </param>
260         /// <returns>
261         /// <para>The bool</para>
262         /// <para></para>
263         /// </returns>
264         [MethodImpl(MethodImplOptions.AggressiveInlining)]
265         protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
              ↪  TLink secondSource, TLink secondTarget)
266             => firstTarget > secondTarget || firstTarget == secondTarget && firstSource >
                  ↪  secondSource;
267
268         /// <summary>
269         /// <para>
270         /// Clears the node using the specified node.
271         /// </para>
272         /// <para></para>
273         /// </summary>
274         /// <param name="node">
275         /// <para>The node.</para>
276         /// <para></para>
277         /// </param>
278         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
279         protected override void ClearNode(TLink node)
280         {
281             ref var link = ref LinksIndexParts[node];
282             link.LeftAsTarget = Zero;
283             link.RightAsTarget = Zero;
284             link.SizeAsTarget = Zero;
285         }
286     }
287 }
```

1.68 ./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)
        {
            LinksDataParts = linksDataParts;
            LinksIndexParts = linksIndexParts;
            Header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetZero() => 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(ulong value) => value == 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(ulong first, ulong second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(ulong value) => value > 0UL;

        /// <summary>
        /// <para>
        /// 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.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
        ↪  for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(ulong first, ulong second) => first < second;

        /// <summary>
        /// <para>
        /// Increments the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Increment(ulong value) => ++value;

        /// <summary>
        /// <para>
        /// Decrements the value.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Decrement(ulong value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Add(ulong first, ulong second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Subtract(ulong first, ulong second) => first - second;

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪  ref LinksDataParts[link];

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
```

```
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪ ref LinksIndexParts[link];

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

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

1.69 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSizeBalancedTreeMethodsBase.

```
using System.Runtime.CompilerServices;
using Platform.Data.Doublets.Memory.Split.Generic;
using TLink = System.UInt64;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 internal links size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="InternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe abstract class UInt64InternalLinksSizeBalancedTreeMethodsBase :
    ↪ InternalLinksSizeBalancedTreeMethodsBase<TLink>
    {
```

```
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkDataPart<TLink>* LinksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLinkIndexPart<TLink>* LinksIndexParts;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly LinksHeader<TLink>* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
        ↪  instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected UInt64InternalLinksSizeBalancedTreeMethodsBase(LinksConstants<TLink>
        ↪  constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
        ↪  linksIndexParts, LinksHeader<TLink>* header)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts, (byte*)header)
        {
            LinksDataParts = linksDataParts;
            LinksIndexParts = linksIndexParts;
            Header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetZero() => 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
```

```
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(ulong value) => value == 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(ulong first, ulong second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(ulong value) => value > 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(ulong first, ulong second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <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>
```

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

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(ulong first, ulong second) => first < second;

        /// <summary>
        /// <para>
        /// Increments the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Increment(ulong value) => ++value;

        /// <summary>
        /// <para>
        /// Decrements the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Decrement(ulong value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
```

```
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Add(ulong first, ulong second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Subtract(ulong first, ulong second) => first - second;

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪  ref LinksDataParts[link];

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪  ref LinksIndexParts[link];

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
```

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

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

1.70 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesLinkedListMethods.cs

```
using System.Runtime.CompilerServices;
using TLink = System.UInt64;

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

namespace Platform.Data.Doublets.Memory.Split.Generic
{
    /// <summary>
    /// <para>
    /// Represents the int 64 internal links sources linked list methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLink}"/>
    public unsafe class UInt64InternalLinksSourcesLinkedListMethods :
    ↪ InternalLinksSourcesLinkedListMethods<TLink>
    {
        private readonly RawLinkDataPart<TLink>* _linksDataParts;
        private readonly RawLinkIndexPart<TLink>* _linksIndexParts;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64InternalLinksSourcesLinkedListMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt64InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
        ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
            : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
        {
            _linksDataParts = linksDataParts;
            _linksIndexParts = linksIndexParts;
        }

```

```
        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪ ref _linksDataParts[link];

        /// <summary>
        /// <para>
        /// Gets the link index part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link index part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
        ↪ ref _linksIndexParts[link];
    }
}
```

1.71 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesRecursionlessSizeBalanc

```
using System.Runtime.CompilerServices;
using TLink = System.UInt64;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 internal links sources recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
    ↪ UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪ cref="UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
```

```
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref TLink GetLeftReference(TLink node) => ref LinksIndexParts[node].LeftAsSource;

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

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

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

```

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

```
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetTreeRoot(TLink node) => LinksIndexParts[node].RootAsSource;

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

        /// <summary>
        /// <para>
        /// Gets the key part value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;

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

        /// <summary>
        /// <para>
        /// Searches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

```
263         /// <para>The link</para>
264         /// <para></para>
265         /// </returns>
266         public override TLink Search(TLink source, TLink target) =>
            ↪ SearchCore(GetTreeRoot(source), target);
267     }
268 }
```

1.72 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMetho

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

```
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref TLink GetRightReference(TLink node) => ref
        ↪  LinksIndexParts[node].RightAsSource;

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

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  LinksIndexParts[node].LeftAsSource = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
```

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

```
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Target;

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

        /// <summary>
        /// <para>
        /// Searches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public override TLink Search(TLink source, TLink target) =>
        ↪  SearchCore(GetTreeRoot(source), target);
    }
}
```

1.73 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalanc

```
using System.Runtime.CompilerServices;
using TLink = System.UInt64;

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

namespace Platform.Data.Doublets.Memory.Split.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 internal links targets recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↪  UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksDataParts">
        /// <para>A links data parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="linksIndexParts">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public
        ↪  UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref ulong GetLeftReference(ulong node) => ref
        ↪  LinksIndexParts[node].LeftAsTarget;

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

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

```

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

```
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  LinksIndexParts[node].SizeAsTarget = size;

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

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

        /// <summary>
        /// <para>
        /// Gets the key part value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetKeyPartValue(TLink node) => LinksDataParts[node].Source;

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(TLink node)
        {
            ref var link = ref LinksIndexParts[node];
            link.LeftAsTarget = Zero;
            link.RightAsTarget = Zero;
            link.SizeAsTarget = Zero;
```

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

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

```
1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt64InternalLinksTargetsSizeBalancedTreeMethods :
        ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
            ↪ 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 UInt64InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
            ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
            ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
            ↪ linksIndexParts, header) { }
41
42         /// <summary>
43         /// <para>
44         /// Gets the left reference using the specified node.
45         /// </para>
46         /// <para></para>
47         /// </summary>
```

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

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

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

1.75 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64SplitMemoryLinks.cs

```
1 using System;
2 using System.Runtime.CompilerServices;
3 using Platform.Singletons;
4 using Platform.Memory;
5 using Platform.Data.Doublets.Memory.Split.Generic;
6 using TLink = System.UInt64;
```

```
7
8   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10  namespace Platform.Data.Doublets.Memory.Split.Specific
11  {
12      /// <summary>
13      /// <para>
14      /// Represents the int 64 split memory links.
15      /// </para>
16      /// <para></para>
17      /// </summary>
18      /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
19      public unsafe class UInt64SplitMemoryLinks : SplitMemoryLinksBase<TLink>
20      {
21          private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
22          private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
23          private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
24          private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
25          private LinksHeader<ulong>* _header;
26          private RawLinkDataPart<ulong>* _linksDataParts;
27          private RawLinkIndexPart<ulong>* _linksIndexParts;
28
29          /// <summary>
30          /// <para>
31          /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
32          /// </para>
33          /// <para></para>
34          /// </summary>
35          /// <param name="dataMemory">
36          /// <para>A data memory.</para>
37          /// <para></para>
38          /// </param>
39          /// <param name="indexMemory">
40          /// <para>A index memory.</para>
41          /// <para></para>
42          /// </param>
43          [MethodImpl(MethodImplOptions.AggressiveInlining)]
44          public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪  indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
45
46          /// <summary>
47          /// <para>
48          /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
49          /// </para>
50          /// <para></para>
51          /// </summary>
52          /// <param name="dataMemory">
53          /// <para>A data memory.</para>
54          /// <para></para>
55          /// </param>
56          /// <param name="indexMemory">
57          /// <para>A index memory.</para>
58          /// <para></para>
59          /// </param>
60          /// <param name="memoryReservationStep">
61          /// <para>A memory reservation step.</para>
62          /// <para></para>
63          /// </param>
64          [MethodImpl(MethodImplOptions.AggressiveInlining)]
65          public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪  indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
            ↪  memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
            ↪  IndexTreeType.Default, useLinkedList: true) { }
66
67          /// <summary>
68          /// <para>
69          /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
70          /// </para>
71          /// <para></para>
72          /// </summary>
73          /// <param name="dataMemory">
74          /// <para>A data memory.</para>
75          /// <para></para>
76          /// </param>
77          /// <param name="indexMemory">
78          /// <para>A index memory.</para>
79          /// <para></para>
80          /// </param>
81          /// <param name="memoryReservationStep">
```

```
        /// <para>A memory reservation step.</para>
        /// <para></para>
        /// </param>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪  indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
        ↪  this(dataMemory, indexMemory, memoryReservationStep, constants,
        ↪  IndexTreeType.Default, useLinkedList: true) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="dataMemory">
        /// <para>A data memory.</para>
        /// <para></para>
        /// </param>
        /// <param name="indexMemory">
        /// <para>A index memory.</para>
        /// <para></para>
        /// </param>
        /// <param name="memoryReservationStep">
        /// <para>A memory reservation step.</para>
        /// <para></para>
        /// </param>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="indexTreeType">
        /// <para>A index tree type.</para>
        /// <para></para>
        /// </param>
        /// <param name="useLinkedList">
        /// <para>A use linked list.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
        ↪  indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
        ↪  IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
        ↪  memoryReservationStep, constants, useLinkedList)
        {
            if (indexTreeType == IndexTreeType.SizeBalancedTree)
            {
                _createInternalSourceTreeMethods = () => new
                ↪  UInt64InternalLinksSourcesSizeBalancedTreeMethods(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createExternalSourceTreeMethods = () => new
                ↪  UInt64ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createInternalTargetTreeMethods = () => new
                ↪  UInt64InternalLinksTargetsSizeBalancedTreeMethods(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createExternalTargetTreeMethods = () => new
                ↪  UInt64ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
            }
            else
            {
                _createInternalSourceTreeMethods = () => new
                ↪  UInt64InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createExternalSourceTreeMethods = () => new
                ↪  UInt64ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
                _createInternalTargetTreeMethods = () => new
                ↪  UInt64InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
                ↪  _linksDataParts, _linksIndexParts, _header);
```

```
137                 _createExternalTargetTreeMethods = () => new
                    ↪  UInt64ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪  _linksDataParts, _linksIndexParts, _header);
138             }
139             Init(dataMemory, indexMemory);
140         }
141
142         /// <summary>
143         /// <para>
144         /// Sets the pointers using the specified data memory.
145         /// </para>
146         /// <para></para>
147         /// </summary>
148         /// <param name="dataMemory">
149         /// <para>The data memory.</para>
150         /// <para></para>
151         /// </param>
152         /// <param name="indexMemory">
153         /// <para>The index memory.</para>
154         /// <para></para>
155         /// </param>
156         [MethodImpl(MethodImplOptions.AggressiveInlining)]
157         protected override void SetPointers(IResizableDirectMemory dataMemory,
            ↪  IResizableDirectMemory indexMemory)
158         {
159             _linksDataParts = (RawLinkDataPart<TLink>*)dataMemory.Pointer;
160             _linksIndexParts = (RawLinkIndexPart<TLink>*)indexMemory.Pointer;
161             _header = (LinksHeader<TLink>*)indexMemory.Pointer;
162             if (_useLinkedList)
163             {
164                 InternalSourcesListMethods = new
                    ↪  UInt64InternalLinksSourcesLinkedListMethods(Constants, _linksDataParts,
                    ↪  _linksIndexParts);
165             }
166             else
167             {
168                 InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
169             }
170             ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
171             InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
172             ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
173             UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_linksDataParts, _header);
174         }
175
176         /// <summary>
177         /// <para>
178         /// Resets the pointers.
179         /// </para>
180         /// <para></para>
181         /// </summary>
182         [MethodImpl(MethodImplOptions.AggressiveInlining)]
183         protected override void ResetPointers()
184         {
185             base.ResetPointers();
186             _linksDataParts = null;
187             _linksIndexParts = null;
188             _header = null;
189         }
190
191         /// <summary>
192         /// <para>
193         /// Gets the header reference.
194         /// </para>
195         /// <para></para>
196         /// </summary>
197         /// <returns>
198         /// <para>A ref links header of t link</para>
199         /// <para></para>
200         /// </returns>
201         [MethodImpl(MethodImplOptions.AggressiveInlining)]
202         protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
203
204         /// <summary>
205         /// <para>
206         /// Gets the link data part reference using the specified link index.
207         /// </para>
208         /// <para></para>
209         /// </summary>
```

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

```
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(ulong first, ulong second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetZero() => OUL;

        /// <summary>
        /// <para>
        /// Gets the one.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
```

```
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetOne() => 1UL;

        /// <summary>
        /// <para>
        /// Converts the to int 64 using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The long</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override long ConvertToInt64(ulong value) => (long)value;

        /// <summary>
        /// <para>
        /// Converts the to address using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong ConvertToAddress(long value) => (ulong)value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Add(ulong first, ulong second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
```

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

1.76 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs

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

```
            _links = links;
            _header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the link data part reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link data part of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
        ↪  ref _links[link];

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
    }
}
```

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

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Methods.Trees;
using Platform.Converters;
using Platform.Numbers;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the links avl balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="SizedAndThreadedAVLBalancedTreeMethods{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class LinksAvlBalancedTreeMethodsBase<TLink> :
    ↪  SizedAndThreadedAVLBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
    {
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;
        private static readonly UncheckedConverter<TLink, int> _addressToInt32Converter =
        ↪  UncheckedConverter<TLink, int>.Default;
        private static readonly UncheckedConverter<bool, TLink> _boolToAddressConverter =
        ↪  UncheckedConverter<bool, TLink>.Default;
        private static readonly UncheckedConverter<TLink, bool> _addressToBoolConverter =
        ↪  UncheckedConverter<TLink, bool>.Default;
        private static readonly UncheckedConverter<int, TLink> _int32ToAddressConverter =
        ↪  UncheckedConverter<int, TLink>.Default;

        /// <summary>
        /// <para>
        /// The break.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
        protected readonly TLink Break;
        /// <summary>
        /// <para>
        /// The continue.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Continue;
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* Links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksAvlBalancedTreeMethodsBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected LinksAvlBalancedTreeMethodsBase(LinksConstants<TLink> constants, byte* links,
        ↪  byte* header)
        {
            Links = links;
            Header = header;
            Break = constants.Break;
            Continue = constants.Continue;
        }

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract TLink GetTreeRoot();

        /// <summary>
        /// <para>
        /// Gets the base part value using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
```

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

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

```
263         /// <para></para>
264         /// </param>
265         /// <returns>
266         /// <para>The bool</para>
267         /// <para></para>
268         /// </returns>
269         [MethodImpl(MethodImplOptions.AggressiveInlining)]
270         protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
271         {
272             ref var firstLink = ref GetLinkReference(first);
273             ref var secondLink = ref GetLinkReference(second);
274             return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
                ↪  secondLink.Source, secondLink.Target);
275         }
276 
277         /// <summary>
278         /// <para>
279         /// Gets the size value using the specified 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 link</para>
289         /// <para></para>
290         /// </returns>
291         [MethodImpl(MethodImplOptions.AggressiveInlining)]
292         protected virtual TLink GetSizeValue(TLink value) => Bit<TLink>.PartialRead(value, 5,
                ↪  -5);
293 
294         /// <summary>
295         /// <para>
296         /// Sets the size value using the specified stored value.
297         /// </para>
298         /// <para></para>
299         /// </summary>
300         /// <param name="storedValue">
301         /// <para>The stored value.</para>
302         /// <para></para>
303         /// </param>
304         /// <param name="size">
305         /// <para>The size.</para>
306         /// <para></para>
307         /// </param>
308         [MethodImpl(MethodImplOptions.AggressiveInlining)]
309         protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
                ↪  Bit<TLink>.PartialWrite(storedValue, size, 5, -5);
310 
311         /// <summary>
312         /// <para>
313         /// Determines whether this instance get left is child value.
314         /// </para>
315         /// <para></para>
316         /// </summary>
317         /// <param name="value">
318         /// <para>The value.</para>
319         /// <para></para>
320         /// </param>
321         /// <returns>
322         /// <para>The bool</para>
323         /// <para></para>
324         /// </returns>
325         [MethodImpl(MethodImplOptions.AggressiveInlining)]
326         protected virtual bool GetLeftIsChildValue(TLink value)
327         {
328             unchecked
329             {
330                 return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 4, 1));
331                 //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 4, 1), default);
332             }
333         }
334 
335         /// <summary>
336         /// <para>
337         /// Sets the left is child value using the specified stored value.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storedValue">
        /// <para>The stored value.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual void SetLeftIsChildValue(ref TLink storedValue, bool value)
        {
            unchecked
            {
                var previousValue = storedValue;
                var modified = Bit<TLink>.PartialWrite(previousValue,
                ↪  _boolToAddressConverter.Convert(value), 4, 1);
                storedValue = modified;
            }
        }

        /// <summary>
        /// <para>
        /// Determines whether this instance get right is child value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual bool GetRightIsChildValue(TLink value)
        {
            unchecked
            {
                return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 3, 1));
                //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 3, 1), default);
            }
        }

        /// <summary>
        /// <para>
        /// Sets the right is child value using the specified stored value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storedValue">
        /// <para>The stored value.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual void SetRightIsChildValue(ref TLink storedValue, bool value)
        {
            unchecked
            {
                var previousValue = storedValue;
                var modified = Bit<TLink>.PartialWrite(previousValue,
                ↪  _boolToAddressConverter.Convert(value), 3, 1);
                storedValue = modified;
            }
        }

        /// <summary>
        /// <para>
        /// Determines whether this instance is child.
        /// </para>
        /// <para></para>
```

```
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <param name="possibleChild">
        /// <para>The possible child.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected bool IsChild(TLink parent, TLink possibleChild)
        {
            var parentSize = GetSize(parent);
            var childSize = GetSizeOrZero(possibleChild);
            return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
        }

        /// <summary>
        /// <para>
        /// Gets the balance value using the specified stored value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storedValue">
        /// <para>The stored value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The sbyte</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual sbyte GetBalanceValue(TLink storedValue)
        {
            unchecked
            {
                var value = _addressToInt32Converter.Convert(Bit<TLink>.PartialRead(storedValue,
                ↪  0, 3));
                value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
                ↪  end of sbyte
                return (sbyte)value;
            }
        }

        /// <summary>
        /// <para>
        /// Sets the balance value using the specified stored value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storedValue">
        /// <para>The stored value.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual void SetBalanceValue(ref TLink storedValue, sbyte value)
        {
            unchecked
            {
                var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
                ↪  value & 3);
                var modified = Bit<TLink>.PartialWrite(storedValue, packagedValue, 0, 3);
                storedValue = modified;
            }
        }

        /// <summary>
        /// <para>
        /// The zero.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        public TLink this[TLink index]
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get
            {
                var root = GetTreeRoot();
                if (GreaterOrEqualThan(index, GetSize(root)))
                {
                    return Zero;
                }
                while (!EqualToZero(root))
                {
                    var left = GetLeftOrDefault(root);
                    var leftSize = GetSizeOrZero(left);
                    if (LessThan(index, leftSize))
                    {
                        root = left;
                        continue;
                    }
                    if (AreEqual(index, leftSize))
                    {
                        return root;
                    }
                    root = GetRightOrDefault(root);
                    index = Subtract(index, Increment(leftSize));
                }
                return Zero; // TODO: Impossible situation exception (only if tree structure
                ↪  broken)
            }
        }

        /// <summary>
        /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
        ↪  (концом).
        /// </summary>
        /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
        /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
        /// <returns>Индекс искомой связи.</returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink Search(TLink source, TLink target)
        {
            var root = GetTreeRoot();
            while (!EqualToZero(root))
            {
                ref var rootLink = ref GetLinkReference(root);
                var rootSource = rootLink.Source;
                var rootTarget = rootLink.Target;
                if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key < root.Key
                {
                    root = GetLeftOrDefault(root);
                }
                else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key > root.Key
                {
                    root = GetRightOrDefault(root);
                }
                else // node.Key == root.Key
                {
                    return root;
                }
            }
            return Zero;
        }

        // TODO: Return indices range instead of references count
        /// <summary>
        /// <para>
        /// Counts the usages using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
```

```
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink CountUsages(TLink link)
        {
            var root = GetTreeRoot();
            var total = GetSize(root);
            var totalRightIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (LessOrEqualThan(@base, link))
                {
                    root = GetRightOrDefault(root);
                }
                else
                {
                    totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
                    root = GetLeftOrDefault(root);
                }
            }
            root = GetTreeRoot();
            var totalLeftIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (GreaterOrEqualThan(@base, link))
                {
                    root = GetLeftOrDefault(root);
                }
                else
                {
                    totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));

                    root = GetRightOrDefault(root);
                }
            }
            return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
        }

        /// <summary>
        /// <para>
        /// Eaches the usage using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The continue.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink EachUsage(TLink link, Func<IList<TLink>, TLink> handler)
        {
            var root = GetTreeRoot();
            if (EqualToZero(root))
            {
                return Continue;
            }
            TLink first = Zero, current = root;
            while (!EqualToZero(current))
            {
                var @base = GetBasePartValue(current);
                if (GreaterOrEqualThan(@base, link))
                {
                    if (AreEqual(@base, link))
                    {
                        first = current;
                    }
```

```
                    current = GetLeftOrDefault(current);
                }
                else
                {
                    current = GetRightOrDefault(current);
                }
            }
            if (!EqualToZero(first))
            {
                current = first;
                while (true)
                {
                    if (AreEqual(handler(GetLinkValues(current)), Break))
                    {
                        return Break;
                    }
                    current = GetNext(current);
                    if (EqualToZero(current) || !AreEqual(GetBasePartValue(current), link))
                    {
                        break;
                    }
                }
            }
            return Continue;
        }

        /// <summary>
        /// <para>
        /// Prints the node value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="sb">
        /// <para>The sb.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void PrintNodeValue(TLink node, StringBuilder sb)
        {
            ref var link = ref GetLinkReference(node);
            sb.Append(' ');
            sb.Append(link.Source);
            sb.Append('-');
            sb.Append('>');
            sb.Append(link.Target);
        }
    }
}
```

1.78 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksRecursionlessSizeBalancedTreeMethodsBase

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Methods.Trees;
using Platform.Converters;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the links recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="RecursionlessSizeBalancedTreeMethods{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class LinksRecursionlessSizeBalancedTreeMethodsBase<TLink> :
    ↪ RecursionlessSizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
    {
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪ UncheckedConverter<TLink, long>.Default;
```

```
24
25          /// <summary>
26          /// <para>
27          /// The break.
28          /// </para>
29          /// <para></para>
30          /// </summary>
31          protected readonly TLink Break;
32          /// <summary>
33          /// <para>
34          /// The continue.
35          /// </para>
36          /// <para></para>
37          /// </summary>
38          protected readonly TLink Continue;
39          /// <summary>
40          /// <para>
41          /// The links.
42          /// </para>
43          /// <para></para>
44          /// </summary>
45          protected readonly byte* Links;
46          /// <summary>
47          /// <para>
48          /// The header.
49          /// </para>
50          /// <para></para>
51          /// </summary>
52          protected readonly byte* Header;
53
54          /// <summary>
55          /// <para>
56          /// Initializes a new <see cref="LinksRecursionlessSizeBalancedTreeMethodsBase"/>
            ↪  instance.
57          /// </para>
58          /// <para></para>
59          /// </summary>
60          /// <param name="constants">
61          /// <para>A constants.</para>
62          /// <para></para>
63          /// </param>
64          /// <param name="links">
65          /// <para>A links.</para>
66          /// <para></para>
67          /// </param>
68          /// <param name="header">
69          /// <para>A header.</para>
70          /// <para></para>
71          /// </param>
72          [MethodImpl(MethodImplOptions.AggressiveInlining)]
73          protected LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<TLink> constants,
            ↪  byte* links, byte* header)
74          {
75              Links = links;
76              Header = header;
77              Break = constants.Break;
78              Continue = constants.Continue;
79          }
80
81          /// <summary>
82          /// <para>
83          /// Gets the tree root.
84          /// </para>
85          /// <para></para>
86          /// </summary>
87          /// <returns>
88          /// <para>The link</para>
89          /// <para></para>
90          /// </returns>
91          [MethodImpl(MethodImplOptions.AggressiveInlining)]
92          protected abstract TLink GetTreeRoot();
93
94          /// <summary>
95          /// <para>
96          /// Gets the base part value using the specified link.
97          /// </para>
98          /// <para></para>
99          /// </summary>
```

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

```
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪  AsRef<LinksHeader<TLink>>(Header);

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
        ↪  AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
        ↪  _addressToInt64Converter.Convert(link)));

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

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
        {
            ref var firstLink = ref GetLinkReference(first);
            ref var secondLink = ref GetLinkReference(second);
            return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
            ↪  secondLink.Source, secondLink.Target);
        }

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the right of second.
        /// </para>
```

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

```
                {
                    root = GetLeftOrDefault(root);
                }
                else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key > root.Key
                {
                    root = GetRightOrDefault(root);
                }
                else // node.Key == root.Key
                {
                    return root;
                }
            }
            return Zero;
        }

        // TODO: Return indices range instead of references count
        /// <summary>
        /// <para>
        /// Counts the usages using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink CountUsages(TLink link)
        {
            var root = GetTreeRoot();
            var total = GetSize(root);
            var totalRightIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (LessOrEqualThan(@base, link))
                {
                    root = GetRightOrDefault(root);
                }
                else
                {
                    totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
                    root = GetLeftOrDefault(root);
                }
            }
            root = GetTreeRoot();
            var totalLeftIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (GreaterOrEqualThan(@base, link))
                {
                    root = GetLeftOrDefault(root);
                }
                else
                {
                    totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
                    root = GetRightOrDefault(root);
                }
            }
            return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
        }

        /// <summary>
        /// <para>
        /// Eaches the usage using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
```

```
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
        ↪  EachUsageCore(@base, GetTreeRoot(), handler);

        // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
        ↪  low-level MSIL stack.
        /// <summary>
        /// <para>
        /// Eaches the usage core using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The continue.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
        {
            var @continue = Continue;
            if (EqualToZero(link))
            {
                return @continue;
            }
            var linkBasePart = GetBasePartValue(link);
            var @break = Break;
            if (GreaterThan(linkBasePart, @base))
            {
                if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            else if (LessThan(linkBasePart, @base))
            {
                if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            else //if (linkBasePart == @base)
            {
                if (AreEqual(handler(GetLinkValues(link)), @break))
                {
                    return @break;
                }
                if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
                {
                    return @break;
                }
                if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            return @continue;
        }

```

```
        /// <summary>
        /// <para>
        /// Prints the node value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="sb">
        /// <para>The sb.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void PrintNodeValue(TLink node, StringBuilder sb)
        {
            ref var link = ref GetLinkReference(node);
            sb.Append(' ');
            sb.Append(link.Source);
            sb.Append('-');
            sb.Append('>');
            sb.Append(link.Target);
        }
    }
}
```

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

```
using System;
using System.Text;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Methods.Trees;
using Platform.Converters;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the links size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="SizeBalancedTreeMethods{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class LinksSizeBalancedTreeMethodsBase<TLink> :
    ↪  SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
    {
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;

        /// <summary>
        /// <para>
        /// The break.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Break;
        /// <summary>
        /// <para>
        /// The continue.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly TLink Continue;
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly byte* Links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
```

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

```
        /// <para></para>
        /// </param>
        /// <param name="rootTarget">
        /// <para>The root target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract bool FirstIsToTheRightOfSecond(TLink source, TLink target, TLink
        ↪  rootSource, TLink rootTarget);

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <param name="rootSource">
        /// <para>The root source.</para>
        /// <para></para>
        /// </param>
        /// <param name="rootTarget">
        /// <para>The root target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected abstract bool FirstIsToTheLeftOfSecond(TLink source, TLink target, TLink
        ↪  rootSource, TLink rootTarget);

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪  AsRef<LinksHeader<TLink>>(Header);

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
        ↪  AsRef<RawLink<TLink>>(Links + (RawLink<TLink>.SizeInBytes *
        ↪  _addressToInt64Converter.Convert(link)));

```

```
199         /// <summary>
200         /// <para>
201         /// Gets the link values using the specified link index.
202         /// </para>
203         /// <para></para>
204         /// </summary>
205         /// <param name="linkIndex">
206         /// <para>The link index.</para>
207         /// <para></para>
208         /// </param>
209         /// <returns>
210         /// <para>A list of t link</para>
211         /// <para></para>
212         /// </returns>
213         [MethodImpl(MethodImplOptions.AggressiveInlining)]
214         protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
215         {
216             ref var link = ref GetLinkReference(linkIndex);
217             return new Link<TLink>(linkIndex, link.Source, link.Target);
218         }
219
220         /// <summary>
221         /// <para>
222         /// Determines whether this instance first is to the left of second.
223         /// </para>
224         /// <para></para>
225         /// </summary>
226         /// <param name="first">
227         /// <para>The first.</para>
228         /// <para></para>
229         /// </param>
230         /// <param name="second">
231         /// <para>The second.</para>
232         /// <para></para>
233         /// </param>
234         /// <returns>
235         /// <para>The bool</para>
236         /// <para></para>
237         /// </returns>
238         [MethodImpl(MethodImplOptions.AggressiveInlining)]
239         protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
240         {
241             ref var firstLink = ref GetLinkReference(first);
242             ref var secondLink = ref GetLinkReference(second);
243             return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
                 ↪  secondLink.Source, secondLink.Target);
244         }
245
246         /// <summary>
247         /// <para>
248         /// Determines whether this instance first is to the right of second.
249         /// </para>
250         /// <para></para>
251         /// </summary>
252         /// <param name="first">
253         /// <para>The first.</para>
254         /// <para></para>
255         /// </param>
256         /// <param name="second">
257         /// <para>The second.</para>
258         /// <para></para>
259         /// </param>
260         /// <returns>
261         /// <para>The bool</para>
262         /// <para></para>
263         /// </returns>
264         [MethodImpl(MethodImplOptions.AggressiveInlining)]
265         protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
266         {
267             ref var firstLink = ref GetLinkReference(first);
268             ref var secondLink = ref GetLinkReference(second);
269             return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
                 ↪  secondLink.Source, secondLink.Target);
270         }
271
272         /// <summary>
273         /// <para>
274         /// The zero.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink this[TLink index]
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get
            {
                var root = GetTreeRoot();
                if (GreaterOrEqualThan(index, GetSize(root)))
                {
                    return Zero;
                }
                while (!EqualToZero(root))
                {
                    var left = GetLeftOrDefault(root);
                    var leftSize = GetSizeOrZero(left);
                    if (LessThan(index, leftSize))
                    {
                        root = left;
                        continue;
                    }
                    if (AreEqual(index, leftSize))
                    {
                        return root;
                    }
                    root = GetRightOrDefault(root);
                    index = Subtract(index, Increment(leftSize));
                }
                return Zero; // TODO: Impossible situation exception (only if tree structure
                ↪  broken)
            }
        }

        /// <summary>
        /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
        ↪  (концом).
        /// </summary>
        /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
        /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
        /// <returns>Индекс искомой связи.</returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink Search(TLink source, TLink target)
        {
            var root = GetTreeRoot();
            while (!EqualToZero(root))
            {
                ref var rootLink = ref GetLinkReference(root);
                var rootSource = rootLink.Source;
                var rootTarget = rootLink.Target;
                if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key < root.Key
                {
                    root = GetLeftOrDefault(root);
                }
                else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
                ↪  node.Key > root.Key
                {
                    root = GetRightOrDefault(root);
                }
                else // node.Key == root.Key
                {
                    return root;
                }
            }
            return Zero;
        }

        // TODO: Return indices range instead of references count
        /// <summary>
        /// <para>
        /// Counts the usages using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
```

```
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink CountUsages(TLink link)
        {
            var root = GetTreeRoot();
            var total = GetSize(root);
            var totalRightIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (LessOrEqualThan(@base, link))
                {
                    root = GetRightOrDefault(root);
                }
                else
                {
                    totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
                    root = GetLeftOrDefault(root);
                }
            }
            root = GetTreeRoot();
            var totalLeftIgnore = Zero;
            while (!EqualToZero(root))
            {
                var @base = GetBasePartValue(root);
                if (GreaterOrEqualThan(@base, link))
                {
                    root = GetLeftOrDefault(root);
                }
                else
                {
                    totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
                    root = GetRightOrDefault(root);
                }
            }
            return Subtract(Subtract(total, totalRightIgnore), totalLeftIgnore);
        }

        /// <summary>
        /// <para>
        /// Eaches the usage using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
        ↪  EachUsageCore(@base, GetTreeRoot(), handler);

        // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
        ↪  low-level MSIL stack.
        /// <summary>
        /// <para>
        /// Eaches the usage core using the specified base.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@base">
        /// <para>The base.</para>
        /// <para></para>
        /// </param>
        /// <param name="link">
        /// <para>The link.</para>
```

```
        /// <para></para>
        /// </param>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The continue.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
        {
            var @continue = Continue;
            if (EqualToZero(link))
            {
                return @continue;
            }
            var linkBasePart = GetBasePartValue(link);
            var @break = Break;
            if (GreaterThan(linkBasePart, @base))
            {
                if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            else if (LessThan(linkBasePart, @base))
            {
                if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            else //if (linkBasePart == @base)
            {
                if (AreEqual(handler(GetLinkValues(link)), @break))
                {
                    return @break;
                }
                if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
                {
                    return @break;
                }
                if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
                {
                    return @break;
                }
            }
            return @continue;
        }

        /// <summary>
        /// <para>
        /// Prints the node value using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="sb">
        /// <para>The sb.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void PrintNodeValue(TLink node, StringBuilder sb)
        {
            ref var link = ref GetLinkReference(node);
            sb.Append(' ');
            sb.Append(link.Source);
            sb.Append('-');
            sb.Append('>');
            sb.Append(link.Target);
        }
    }
}
```

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the links sources avl balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksAvlBalancedTreeMethodsBase{TLink}"/>
    public unsafe class LinksSourcesAvlBalancedTreeMethods<TLink> :
    ↪ LinksAvlBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksSourcesAvlBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksSourcesAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
        ↪ byte* header) : base(constants, links, header) { }

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

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

        /// <summary>
        /// <para>
        /// Gets the left using the specified node.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  GetLinkReference(node).LeftAsSource = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  GetLinkReference(node).RightAsSource = right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

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

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

```
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetBasePartValue(TLink link) => GetLinkReference(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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
        ↪  (AreEqual(firstSource, secondSource) && LessThan(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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
        ↪  (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
```

```
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(TLink node)
        {
            ref var link = ref GetLinkReference(node);
            link.LeftAsSource = Zero;
            link.RightAsSource = Zero;
            link.SizeAsSource = Zero;
        }
    }
}
```

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the links sources recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe class LinksSourcesRecursionlessSizeBalancedTreeMethods<TLink> :
    ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksSourcesRecursionlessSizeBalancedTreeMethods"/>
        ↪ instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
        ↪ byte* links, byte* header) : base(constants, links, header) { }

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

        /// <summary>
        /// <para>
        /// Gets the right reference using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
```

```
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref TLink GetRightReference(TLink node) => ref
        ↪  GetLinkReference(node).RightAsSource;

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

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  GetLinkReference(node).LeftAsSource = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [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) =>
            ↪ GetLinkReference(node).SizeAsSource = size;
172
173         /// <summary>
174         /// <para>
175         /// Gets the tree root.
176         /// </para>
177         /// <para></para>
178         /// </summary>
179         /// <returns>
180         /// <para>The link</para>
181         /// <para></para>
182         /// </returns>
183         [MethodImpl(MethodImplOptions.AggressiveInlining)]
184         protected override TLink GetTreeRoot() => GetHeaderReference().RootAsSource;
185
186         /// <summary>
187         /// <para>
188         /// Gets the base part value using the specified link.
189         /// </para>
190         /// <para></para>
191         /// </summary>
192         /// <param name="link">
193         /// <para>The link.</para>
194         /// <para></para>
195         /// </param>
196         /// <returns>
197         /// <para>The link</para>
198         /// <para></para>
199         /// </returns>
200         [MethodImpl(MethodImplOptions.AggressiveInlining)]
201         protected override TLink GetBasePartValue(TLink link) => GetLinkReference(link).Source;
202
203         /// <summary>
204         /// <para>
205         /// Determines whether this instance first is to the left of second.
206         /// </para>
207         /// <para></para>
208         /// </summary>
209         /// <param name="firstSource">
210         /// <para>The first source.</para>
211         /// <para></para>
212         /// </param>
```

```
        /// <param name="firstTarget">
        /// <para>The first target.</para>
        /// <para></para>
        /// </param>
        /// <param name="secondSource">
        /// <para>The second source.</para>
        /// <para></para>
        /// </param>
        /// <param name="secondTarget">
        /// <para>The second target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheLeftOfSecond(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget) => LessThan(firstSource, secondSource) ||
        ↪  (AreEqual(firstSource, secondSource) && LessThan(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(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget) => GreaterThan(firstSource, secondSource) ||
        ↪  (AreEqual(firstSource, secondSource) && GreaterThan(firstTarget, secondTarget));

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

1.82 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksSourcesSizeBalancedTreeMethods.cs

```
using System.Runtime.CompilerServices;

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

```

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

```
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetLeft(TLink node) => GetLinkReference(node).LeftAsSource;

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  GetLinkReference(node).LeftAsSource = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  GetLinkReference(node).RightAsSource = right;

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

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

1.83 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.cs

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

```
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksTargetsAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
        ↪  byte* header) : base(constants, links, header) { }

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

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

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

        /// <summary>
        /// <para>
        /// Gets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
```

```
96          /// <para></para>
97          /// </param>
98          /// <returns>
99          /// <para>The link</para>
100         /// <para></para>
101         /// </returns>
102         [MethodImpl(MethodImplOptions.AggressiveInlining)]
103         protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsTarget;
104
105         /// <summary>
106         /// <para>
107         /// Sets the left using the specified node.
108         /// </para>
109         /// <para></para>
110         /// </summary>
111         /// <param name="node">
112         /// <para>The node.</para>
113         /// <para></para>
114         /// </param>
115         /// <param name="left">
116         /// <para>The left.</para>
117         /// <para></para>
118         /// </param>
119         [MethodImpl(MethodImplOptions.AggressiveInlining)]
120         protected override void SetLeft(TLink node, TLink left) =>
            ↪ GetLinkReference(node).LeftAsTarget = left;
121
122         /// <summary>
123         /// <para>
124         /// Sets the right using the specified node.
125         /// </para>
126         /// <para></para>
127         /// </summary>
128         /// <param name="node">
129         /// <para>The node.</para>
130         /// <para></para>
131         /// </param>
132         /// <param name="right">
133         /// <para>The right.</para>
134         /// <para></para>
135         /// </param>
136         [MethodImpl(MethodImplOptions.AggressiveInlining)]
137         protected override void SetRight(TLink node, TLink right) =>
            ↪ GetLinkReference(node).RightAsTarget = right;
138
139         /// <summary>
140         /// <para>
141         /// Gets the size using the specified node.
142         /// </para>
143         /// <para></para>
144         /// </summary>
145         /// <param name="node">
146         /// <para>The node.</para>
147         /// <para></para>
148         /// </param>
149         /// <returns>
150         /// <para>The link</para>
151         /// <para></para>
152         /// </returns>
153         [MethodImpl(MethodImplOptions.AggressiveInlining)]
154         protected override TLink GetSize(TLink node) =>
            ↪ GetSizeValue(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) => SetSizeValue(ref
            ↪ GetLinkReference(node).SizeAsTarget, size);
172
173         /// <summary>
174         /// <para>
175         /// Determines whether this instance get left is child.
176         /// </para>
177         /// <para></para>
178         /// </summary>
179         /// <param name="node">
180         /// <para>The node.</para>
181         /// <para></para>
182         /// </param>
183         /// <returns>
184         /// <para>The bool</para>
185         /// <para></para>
186         /// </returns>
187         [MethodImpl(MethodImplOptions.AggressiveInlining)]
188         protected override bool GetLeftIsChild(TLink node) =>
            ↪ GetLeftIsChildValue(GetLinkReference(node).SizeAsTarget);
189
190         /// <summary>
191         /// <para>
192         /// Sets the left is child using the specified node.
193         /// </para>
194         /// <para></para>
195         /// </summary>
196         /// <param name="node">
197         /// <para>The node.</para>
198         /// <para></para>
199         /// </param>
200         /// <param name="value">
201         /// <para>The value.</para>
202         /// <para></para>
203         /// </param>
204         [MethodImpl(MethodImplOptions.AggressiveInlining)]
205         protected override void SetLeftIsChild(TLink node, bool value) =>
            ↪ SetLeftIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
206
207         /// <summary>
208         /// <para>
209         /// Determines whether this instance get right is child.
210         /// </para>
211         /// <para></para>
212         /// </summary>
213         /// <param name="node">
214         /// <para>The node.</para>
215         /// <para></para>
216         /// </param>
217         /// <returns>
218         /// <para>The bool</para>
219         /// <para></para>
220         /// </returns>
221         [MethodImpl(MethodImplOptions.AggressiveInlining)]
222         protected override bool GetRightIsChild(TLink node) =>
            ↪ GetRightIsChildValue(GetLinkReference(node).SizeAsTarget);
223
224         /// <summary>
225         /// <para>
226         /// Sets the right is child using the specified node.
227         /// </para>
228         /// <para></para>
229         /// </summary>
230         /// <param name="node">
231         /// <para>The node.</para>
232         /// <para></para>
233         /// </param>
234         /// <param name="value">
235         /// <para>The value.</para>
236         /// <para></para>
237         /// </param>
238         [MethodImpl(MethodImplOptions.AggressiveInlining)]
239         protected override void SetRightIsChild(TLink node, bool value) =>
            ↪ SetRightIsChildValue(ref GetLinkReference(node).SizeAsTarget, value);
240
241         /// <summary>
242         /// <para>
```

```
        /// Gets the balance using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The sbyte</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override sbyte GetBalance(TLink node) =>
        ↪  GetBalanceValue(GetLinkReference(node).SizeAsTarget);

        /// <summary>
        /// <para>
        /// Sets the balance using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetBalance(TLink node, sbyte value) => SetBalanceValue(ref
        ↪  GetLinkReference(node).SizeAsTarget, value);

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

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

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

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

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

```
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>
```

```
    /// Represents the links targets recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe class LinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
    ↪  LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksTargetsRecursionlessSizeBalancedTreeMethods"/>
        ↪  instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
        ↪  byte* links, byte* header) : base(constants, links, header) { }

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

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

        /// <summary>
        /// <para>
        /// Gets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

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

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

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(TLink node)
        {
            ref var link = ref GetLinkReference(node);
            link.LeftAsTarget = Zero;
            link.RightAsTarget = Zero;
            link.SizeAsTarget = Zero;
        }
    }
}
```

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the links targets size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe class LinksTargetsSizeBalancedTreeMethods<TLink> :
    ↪  LinksSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksTargetsSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
```

```
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
        ↪  byte* header) : base(constants, links, header) { }

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

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

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

        /// <summary>
        /// <para>
        /// Gets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
```

```
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetRight(TLink node) => GetLinkReference(node).RightAsTarget;

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(TLink node, TLink left) =>
        ↪  GetLinkReference(node).LeftAsTarget = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(TLink node, TLink right) =>
        ↪  GetLinkReference(node).RightAsTarget = right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetSize(TLink node) => GetLinkReference(node).SizeAsTarget;

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink node, TLink size) =>
        ↪  GetLinkReference(node).SizeAsTarget = size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
```

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

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

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

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

```
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheRightOfSecond(TLink firstSource, TLink firstTarget,
        ↪  TLink secondSource, TLink secondTarget) => GreaterThan(firstTarget, secondTarget) ||
        ↪  (AreEqual(firstTarget, secondTarget) && GreaterThan(firstSource, secondSource));

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(TLink node)
        {
            ref var link = ref GetLinkReference(node);
            link.LeftAsTarget = Zero;
            link.RightAsTarget = Zero;
            link.SizeAsTarget = Zero;
        }
    }
}
```

1.86 ./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs

```
using System;
using System.Runtime.CompilerServices;
using Platform.Singletons;
using Platform.Memory;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the united memory links.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UnitedMemoryLinksBase{TLink}"/>
    public unsafe class UnitedMemoryLinks<TLink> : UnitedMemoryLinksBase<TLink>
    {
        private readonly Func<ILinksTreeMethods<TLink>> _createSourceTreeMethods;
        private readonly Func<ILinksTreeMethods<TLink>> _createTargetTreeMethods;
        private byte* _header;
        private byte* _links;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="address">
        /// <para>A address.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }

        /// <summary>
        /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
        ↪  минимальным шагом расширения базы данных.
        /// </summary>
        /// <param name="address">Полный пусть к файлу базы данных.</param>
        /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
        ↪  байтах.</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
44          public UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
            ↪  FileMappedResizableDirectMemory(address, memoryReservationStep),
            ↪  memoryReservationStep) { }
45
46          /// <summary>
47          /// <para>
48          /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
49          /// </para>
50          /// <para></para>
51          /// </summary>
52          /// <param name="memory">
53          /// <para>A memory.</para>
54          /// <para></para>
55          /// </param>
56          [MethodImpl(MethodImplOptions.AggressiveInlining)]
57          public UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
            ↪  DefaultLinksSizeStep) { }
58
59          /// <summary>
60          /// <para>
61          /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
62          /// </para>
63          /// <para></para>
64          /// </summary>
65          /// <param name="memory">
66          /// <para>A memory.</para>
67          /// <para></para>
68          /// </param>
69          /// <param name="memoryReservationStep">
70          /// <para>A memory reservation step.</para>
71          /// <para></para>
72          /// </param>
73          [MethodImpl(MethodImplOptions.AggressiveInlining)]
74          public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep) :
            ↪  this(memory, memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
            ↪  IndexTreeType.Default) { }
75
76          /// <summary>
77          /// <para>
78          /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
79          /// </para>
80          /// <para></para>
81          /// </summary>
82          /// <param name="memory">
83          /// <para>A memory.</para>
84          /// <para></para>
85          /// </param>
86          /// <param name="memoryReservationStep">
87          /// <para>A memory reservation step.</para>
88          /// <para></para>
89          /// </param>
90          /// <param name="constants">
91          /// <para>A constants.</para>
92          /// <para></para>
93          /// </param>
94          /// <param name="indexTreeType">
95          /// <para>A index tree type.</para>
96          /// <para></para>
97          /// </param>
98          [MethodImpl(MethodImplOptions.AggressiveInlining)]
99          public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep,
            ↪  LinksConstants<TLink> constants, IndexTreeType indexTreeType) : base(memory,
            ↪  memoryReservationStep, constants)
100         {
101             if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
102             {
103                 _createSourceTreeMethods = () => new
                    ↪  LinksSourcesAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
104                 _createTargetTreeMethods = () => new
                    ↪  LinksTargetsAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
105             }
106             else
107             {
108                 _createSourceTreeMethods = () => new
                    ↪  LinksSourcesSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
109                 _createTargetTreeMethods = () => new
                    ↪  LinksTargetsSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
```

```
            }
            Init(memory, memoryReservationStep);
        }

        /// <summary>
        /// <para>
        /// Sets the pointers using the specified memory.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="memory">
        /// <para>The memory.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetPointers(IResizableDirectMemory memory)
        {
            _links = (byte*)memory.Pointer;
            _header = _links;
            SourcesTreeMethods = _createSourceTreeMethods();
            TargetsTreeMethods = _createTargetTreeMethods();
            UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_links, _header);
        }

        /// <summary>
        /// <para>
        /// Resets the pointers.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ResetPointers()
        {
            base.ResetPointers();
            _links = null;
            _header = null;
        }

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪  AsRef<LinksHeader<TLink>>(_header);

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link index.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLink<TLink> GetLinkReference(TLink linkIndex) => ref
        ↪  AsRef<RawLink<TLink>>(_links + (LinkSizeInBytes * ConvertToInt64(linkIndex)));
    }
}
```

1.87 ./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinksBase.cs

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Disposables;
using Platform.Singletons;
using Platform.Converters;
```

```
using Platform.Numbers;
using Platform.Memory;
using Platform.Data.Exceptions;

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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the united memory links base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="DisposableBase"/>
    /// <seealso cref="ILinks{TLink}"/>
    public abstract class UnitedMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;
        private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;
        private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
        ↪  UncheckedConverter<long, TLink>.Default;

        private static readonly TLink _zero = default;
        private static readonly TLink _one = Arithmetic.Increment(_zero);

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

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

        /// <summary>
        /// <para>
        /// The link size in bytes.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;

        /// <summary>
        /// <para>
        /// The memory.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly IResizableDirectMemory _memory;
        /// <summary>
        /// <para>
        /// The memory reservation step.
        /// </para>
        /// <para></para>
        /// </summary>
        protected readonly long _memoryReservationStep;

        /// <summary>
        /// <para>
        /// The targets tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        protected ILinksTreeMethods<TLink> TargetsTreeMethods;
        /// <summary>
        /// <para>
        /// The sources tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
84          protected ILinksTreeMethods<TLink> SourcesTreeMethods;
85          // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
            ↪  нужно использовать не список а дерево, так как так можно быстрее проверить на
            ↪  наличие связи внутри
86          /// <summary>
87          /// <para>
88          /// The unused links list methods.
89          /// </para>
90          /// <para></para>
91          /// </summary>
92          protected ILinksListMethods<TLink> UnusedLinksListMethods;
93
94          /// <summary>
95          /// Возвращает общее число связей находящихся в хранилище.
96          /// </summary>
97          protected virtual TLink Total
98          {
99              [MethodImpl(MethodImplOptions.AggressiveInlining)]
100             get
101             {
102                 ref var header = ref GetHeaderReference();
103                 return Subtract(header.AllocatedLinks, header.FreeLinks);
104             }
105         }
106
107         /// <summary>
108         /// <para>
109         /// Gets the constants value.
110         /// </para>
111         /// <para></para>
112         /// </summary>
113         public virtual LinksConstants<TLink> Constants
114         {
115             [MethodImpl(MethodImplOptions.AggressiveInlining)]
116             get;
117         }
118
119         /// <summary>
120         /// <para>
121         /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
122         /// </para>
123         /// <para></para>
124         /// </summary>
125         /// <param name="memory">
126         /// <para>A memory.</para>
127         /// <para></para>
128         /// </param>
129         /// <param name="memoryReservationStep">
130         /// <para>A memory reservation step.</para>
131         /// <para></para>
132         /// </param>
133         /// <param name="constants">
134         /// <para>A constants.</para>
135         /// <para></para>
136         /// </param>
137         [MethodImpl(MethodImplOptions.AggressiveInlining)]
138         protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
            ↪  memoryReservationStep, LinksConstants<TLink> constants)
139         {
140             _memory = memory;
141             _memoryReservationStep = memoryReservationStep;
142             Constants = constants;
143         }
144
145         /// <summary>
146         /// <para>
147         /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
148         /// </para>
149         /// <para></para>
150         /// </summary>
151         /// <param name="memory">
152         /// <para>A memory.</para>
153         /// <para></para>
154         /// </param>
155         /// <param name="memoryReservationStep">
156         /// <para>A memory reservation step.</para>
157         /// <para></para>
158         /// </param>
159         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
160         protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
            ↪ memoryReservationStep) : this(memory, memoryReservationStep,
            ↪ Default<LinksConstants<TLink>>.Instance) { }
161
162         /// <summary>
163         /// <para>
164         /// Inits the memory.
165         /// </para>
166         /// <para></para>
167         /// </summary>
168         /// <param name="memory">
169         /// <para>The memory.</para>
170         /// <para></para>
171         /// </param>
172         /// <param name="memoryReservationStep">
173         /// <para>The memory reservation step.</para>
174         /// <para></para>
175         /// </param>
176         [MethodImpl(MethodImplOptions.AggressiveInlining)]
177         protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
178         {
179             if (memory.ReservedCapacity < memoryReservationStep)
180             {
181                 memory.ReservedCapacity = memoryReservationStep;
182             }
183             SetPointers(memory);
184             ref var header = ref GetHeaderReference();
185             // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
186             memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
                ↪ LinkHeaderSizeInBytes;
187             // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
188             header.ReservedLinks = ConvertToAddress((memory.ReservedCapacity -
                ↪ LinkHeaderSizeInBytes) / LinkSizeInBytes);
189         }
190
191         /// <summary>
192         /// <para>
193         /// Counts the restrictions.
194         /// </para>
195         /// <para></para>
196         /// </summary>
197         /// <param name="restrictions">
198         /// <para>The restrictions.</para>
199         /// <para></para>
200         /// </param>
201         /// <exception cref="NotSupportedException">
202         /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
203         /// <para></para>
204         /// </exception>
205         /// <returns>
206         /// <para>The link</para>
207         /// <para></para>
208         /// </returns>
209         [MethodImpl(MethodImplOptions.AggressiveInlining)]
210         public virtual TLink Count(IList<TLink> restrictions)
211         {
212             // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
213             if (restrictions.Count == 0)
214             {
215                 return Total;
216             }
217             var constants = Constants;
218             var any = constants.Any;
219             var index = restrictions[constants.IndexPart];
220             if (restrictions.Count == 1)
221             {
222                 if (AreEqual(index, any))
223                 {
224                     return Total;
225                 }
226                 return Exists(index) ? GetOne() : GetZero();
227             }
228             if (restrictions.Count == 2)
229             {
230                 var value = restrictions[1];
231                 if (AreEqual(index, any))
232                 {
233                     if (AreEqual(value, any))
```

```
                    {
                        return Total; // Any - как отсутствие ограничения
                    }
                    return Add(SourcesTreeMethods.CountUsages(value),
                    ↪  TargetsTreeMethods.CountUsages(value));
                }
                else
                {
                    if (!Exists(index))
                    {
                        return GetZero();
                    }
                    if (AreEqual(value, any))
                    {
                        return GetOne();
                    }
                    ref var storedLinkValue = ref GetLinkReference(index);
                    if (AreEqual(storedLinkValue.Source, value) ||
                    ↪  AreEqual(storedLinkValue.Target, value))
                    {
                        return GetOne();
                    }
                    return GetZero();
                }
            }
            if (restrictions.Count == 3)
            {
                var source = restrictions[constants.SourcePart];
                var target = restrictions[constants.TargetPart];
                if (AreEqual(index, any))
                {
                    if (AreEqual(source, any) && AreEqual(target, any))
                    {
                        return Total;
                    }
                    else if (AreEqual(source, any))
                    {
                        return TargetsTreeMethods.CountUsages(target);
                    }
                    else if (AreEqual(target, any))
                    {
                        return SourcesTreeMethods.CountUsages(source);
                    }
                    else //if(source != Any && target != Any)
                    {
                        // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
                        var link = SourcesTreeMethods.Search(source, target);
                        return AreEqual(link, constants.Null) ? GetZero() : GetOne();
                    }
                }
                else
                {
                    if (!Exists(index))
                    {
                        return GetZero();
                    }
                    if (AreEqual(source, any) && AreEqual(target, any))
                    {
                        return GetOne();
                    }
                    ref var storedLinkValue = ref GetLinkReference(index);
                    if (!AreEqual(source, any) && !AreEqual(target, any))
                    {
                        if (AreEqual(storedLinkValue.Source, source) &&
                        ↪  AreEqual(storedLinkValue.Target, target))
                        {
                            return GetOne();
                        }
                        return GetZero();
                    }
                    var value = default(TLink);
                    if (AreEqual(source, any))
                    {
                        value = target;
                    }
                    if (AreEqual(target, any))
                    {
                        value = source;
```

```
                    }
                    if (AreEqual(storedLinkValue.Source, value) ||
                        AreEqual(storedLinkValue.Target, value))
                    {
                        return GetOne();
                    }
                    return GetZero();
                }
            }
            throw new NotSupportedException("Другие размеры и способы ограничений не
                поддерживаются.");
        }

        /// <summary>
        /// <para>
        /// Eaches the handler.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="handler">
        /// <para>The handler.</para>
        /// <para></para>
        /// </param>
        /// <param name="restrictions">
        /// <para>The restrictions.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="NotSupportedException">
        /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
        {
            var constants = Constants;
            var @break = constants.Break;
            if (restrictions.Count == 0)
            {
                for (var link = GetOne(); LessOrEqualThan(link,
                    GetHeaderReference().AllocatedLinks); link = Increment(link))
                {
                    if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
                    {
                        return @break;
                    }
                }
                return @break;
            }
            var @continue = constants.Continue;
            var any = constants.Any;
            var index = restrictions[constants.IndexPart];
            if (restrictions.Count == 1)
            {
                if (AreEqual(index, any))
                {
                    return Each(handler, Array.Empty<TLink>());
                }
                if (!Exists(index))
                {
                    return @continue;
                }
                return handler(GetLinkStruct(index));
            }
            if (restrictions.Count == 2)
            {
                var value = restrictions[1];
                if (AreEqual(index, any))
                {
                    if (AreEqual(value, any))
                    {
                        return Each(handler, Array.Empty<TLink>());
                    }
                    if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
                    {
```

```
                    return @break;
                }
                return Each(handler, new Link<TLink>(index, any, value));
            }
            else
            {
                if (!Exists(index))
                {
                    return @continue;
                }
                if (AreEqual(value, any))
                {
                    return handler(GetLinkStruct(index));
                }
                ref var storedLinkValue = ref GetLinkReference(index);
                if (AreEqual(storedLinkValue.Source, value) ||
                    AreEqual(storedLinkValue.Target, value))
                {
                    return handler(GetLinkStruct(index));
                }
                return @continue;
            }
        }
        if (restrictions.Count == 3)
        {
            var source = restrictions[constants.SourcePart];
            var target = restrictions[constants.TargetPart];
            if (AreEqual(index, any))
            {
                if (AreEqual(source, any) && AreEqual(target, any))
                {
                    return Each(handler, Array.Empty<TLink>());
                }
                else if (AreEqual(source, any))
                {
                    return TargetsTreeMethods.EachUsage(target, handler);
                }
                else if (AreEqual(target, any))
                {
                    return SourcesTreeMethods.EachUsage(source, handler);
                }
                else //if(source != Any && target != Any)
                {
                    var link = SourcesTreeMethods.Search(source, target);
                    return AreEqual(link, constants.Null) ? @continue :
                    ↪  handler(GetLinkStruct(link));
                }
            }
            else
            {
                if (!Exists(index))
                {
                    return @continue;
                }
                if (AreEqual(source, any) && AreEqual(target, any))
                {
                    return handler(GetLinkStruct(index));
                }
                ref var storedLinkValue = ref GetLinkReference(index);
                if (!AreEqual(source, any) && !AreEqual(target, any))
                {
                    if (AreEqual(storedLinkValue.Source, source) &&
                        AreEqual(storedLinkValue.Target, target))
                    {
                        return handler(GetLinkStruct(index));
                    }
                    return @continue;
                }
                var value = default(TLink);
                if (AreEqual(source, any))
                {
                    value = target;
                }
                if (AreEqual(target, any))
                {
                    value = source;
                }
                if (AreEqual(storedLinkValue.Source, value) ||
```

```
                        AreEqual(storedLinkValue.Target, value))
                    {
                        return handler(GetLinkStruct(index));
                    }
                    return @continue;
                }
            }
            throw new NotSupportedException("Другие размеры и способы ограничений не
            ↪ поддерживаются.");
        }

        /// <remarks>
        /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
        ↪ в другом месте (но не в менеджере памяти, а в логике Links)
        /// </remarks>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
        {
            var constants = Constants;
            var @null = constants.Null;
            var linkIndex = restrictions[constants.IndexPart];
            ref var link = ref GetLinkReference(linkIndex);
            ref var header = ref GetHeaderReference();
            ref var firstAsSource = ref header.RootAsSource;
            ref var firstAsTarget = ref header.RootAsTarget;
            // Будет корректно работать только в том случае, если пространство выделенной связи
            ↪ предварительно заполнено нулями
            if (!AreEqual(link.Source, @null))
            {
                SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
            }
            if (!AreEqual(link.Target, @null))
            {
                TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
            }
            link.Source = substitution[constants.SourcePart];
            link.Target = substitution[constants.TargetPart];
            if (!AreEqual(link.Source, @null))
            {
                SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
            }
            if (!AreEqual(link.Target, @null))
            {
                TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
            }
            return linkIndex;
        }

        /// <remarks>
        /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
        ↪ пространство
        /// </remarks>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public virtual TLink Create(IList<TLink> restrictions)
        {
            ref var header = ref GetHeaderReference();
            var freeLink = header.FirstFreeLink;
            if (!AreEqual(freeLink, Constants.Null))
            {
                UnusedLinksListMethods.Detach(freeLink);
            }
            else
            {
                var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
                if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
                {
                    throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
                }
                if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
                {
                    _memory.ReservedCapacity += _memoryReservationStep;
                    SetPointers(_memory);
                    header = ref GetHeaderReference();
                    header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
                    ↪ LinkSizeInBytes);
                }
                freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
                _memory.UsedCapacity += LinkSizeInBytes;
            }
```

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

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

```
                return AreEqual(link.SizeAsSource, default) && !AreEqual(link.Source, default);
            }
            else
            {
                return true;
            }
        }

        /// <summary>
        /// <para>
        /// Gets the one.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink GetOne() => _one;

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink GetZero() => default;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual bool AreEqual(TLink first, TLink second) =>
        ↪  _equalityComparer.Equals(first, second);

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
        ↪  second) < 0;

```

```
763         /// <summary>
764         /// <para>
765         /// Determines whether this instance less or equal than.
766         /// </para>
767         /// <para></para>
768         /// </summary>
769         /// <param name="first">
770         /// <para>The first.</para>
771         /// <para></para>
772         /// </param>
773         /// <param name="second">
774         /// <para>The second.</para>
775         /// <para></para>
776         /// </param>
777         /// <returns>
778         /// <para>The bool</para>
779         /// <para></para>
780         /// </returns>
781         [MethodImpl(MethodImplOptions.AggressiveInlining)]
782         protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
              ↪ _comparer.Compare(first, second) <= 0;
783
784         /// <summary>
785         /// <para>
786         /// Determines whether this instance greater than.
787         /// </para>
788         /// <para></para>
789         /// </summary>
790         /// <param name="first">
791         /// <para>The first.</para>
792         /// <para></para>
793         /// </param>
794         /// <param name="second">
795         /// <para>The second.</para>
796         /// <para></para>
797         /// </param>
798         /// <returns>
799         /// <para>The bool</para>
800         /// <para></para>
801         /// </returns>
802         [MethodImpl(MethodImplOptions.AggressiveInlining)]
803         protected virtual bool GreaterThan(TLink first, TLink second) =>
              ↪ _comparer.Compare(first, second) > 0;
804
805         /// <summary>
806         /// <para>
807         /// Determines whether this instance greater or equal than.
808         /// </para>
809         /// <para></para>
810         /// </summary>
811         /// <param name="first">
812         /// <para>The first.</para>
813         /// <para></para>
814         /// </param>
815         /// <param name="second">
816         /// <para>The second.</para>
817         /// <para></para>
818         /// </param>
819         /// <returns>
820         /// <para>The bool</para>
821         /// <para></para>
822         /// </returns>
823         [MethodImpl(MethodImplOptions.AggressiveInlining)]
824         protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
              ↪ _comparer.Compare(first, second) >= 0;
825
826         /// <summary>
827         /// <para>
828         /// Converts the to int 64 using the specified value.
829         /// </para>
830         /// <para></para>
831         /// </summary>
832         /// <param name="value">
833         /// <para>The value.</para>
834         /// <para></para>
835         /// </param>
836         /// <returns>
837         /// <para>The long</para>
```

```
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual long ConvertToInt64(TLink value) =>
        ↪ _addressToInt64Converter.Convert(value);

        /// <summary>
        /// <para>
        /// Converts the to address using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink ConvertToAddress(long value) =>
        ↪ _int64ToAddressConverter.Convert(value);

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
        ↪ second);

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink Subtract(TLink first, TLink second) =>
        ↪ Arithmetic<TLink>.Subtract(first, second);

        /// <summary>
        /// <para>
        /// Increments the link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
```

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

1.88 ./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs

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

```
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="AbsoluteCircularDoublyLinkedListMethods{TLink}"/>
    /// <seealso cref="ILinksListMethods{TLink}"/>
    public unsafe class UnusedLinksListMethods<TLink> :
    ↪  AbsoluteCircularDoublyLinkedListMethods<TLink>, ILinksListMethods<TLink>
    {
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;

        private readonly byte* _links;
        private readonly byte* _header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UnusedLinksListMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UnusedLinksListMethods(byte* links, byte* header)
        {
            _links = links;
            _header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
        ↪  AsRef<LinksHeader<TLink>>(_header);

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual ref RawLink<TLink> GetLinkReference(TLink link) => ref
        ↪  AsRef<RawLink<TLink>>(_links + (RawLink<TLink>.SizeInBytes *
        ↪  _addressToInt64Converter.Convert(link)));

        /// <summary>
        /// <para>
        /// Gets the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
```

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;

        /// <summary>
        /// <para>
        /// Gets the last.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetLast() => GetHeaderReference().LastFreeLink;

        /// <summary>
        /// <para>
        /// Gets the previous using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetPrevious(TLink element) => GetLinkReference(element).Source;

        /// <summary>
        /// <para>
        /// Gets the next using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetNext(TLink element) => GetLinkReference(element).Target;

        /// <summary>
        /// <para>
        /// Gets the size.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetSize() => GetHeaderReference().FreeLinks;

        /// <summary>
        /// <para>
        /// Sets the first using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
        ↪  element;

        /// <summary>
```

```
        /// <para>
        /// Sets the last using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
        ↪  element;

        /// <summary>
        /// <para>
        /// Sets the previous using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        /// <param name="previous">
        /// <para>The previous.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetPrevious(TLink element, TLink previous) =>
        ↪  GetLinkReference(element).Source = previous;

        /// <summary>
        /// <para>
        /// Sets the next using the specified element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        /// <param name="next">
        /// <para>The next.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetNext(TLink element, TLink next) =>
        ↪  GetLinkReference(element).Target = next;

        /// <summary>
        /// <para>
        /// Sets the size using the specified size.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
    }
}
```

1.89 ./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs

```
using Platform.Unsafe;
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United
{
    /// <summary>
    /// <para>
    /// The raw link.
    /// </para>
```

```
    /// <para></para>
    /// </summary>
    public struct RawLink<TLink> : IEquatable<RawLink<TLink>>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// The size.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly long SizeInBytes = Structure<RawLink<TLink>>.Size;

        /// <summary>
        /// <para>
        /// The source.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink Source;
        /// <summary>
        /// <para>
        /// The target.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink Target;
        /// <summary>
        /// <para>
        /// The left as source.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink LeftAsSource;
        /// <summary>
        /// <para>
        /// The right as source.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink RightAsSource;
        /// <summary>
        /// <para>
        /// The size as source.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink SizeAsSource;
        /// <summary>
        /// <para>
        /// The left as target.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink LeftAsTarget;
        /// <summary>
        /// <para>
        /// The right as target.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink RightAsTarget;
        /// <summary>
        /// <para>
        /// The size as target.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink SizeAsTarget;

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="obj">
```

```
        /// <para>The obj.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override bool Equals(object obj) => obj is RawLink<TLink> link ? Equals(link) :
        ↪  false;

        /// <summary>
        /// <para>
        /// Determines whether this instance equals.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
        /// <para>The other.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public bool Equals(RawLink<TLink> other)
            => _equalityComparer.Equals(Source, other.Source)
            && _equalityComparer.Equals(Target, other.Target)
            && _equalityComparer.Equals(LeftAsSource, other.LeftAsSource)
            && _equalityComparer.Equals(RightAsSource, other.RightAsSource)
            && _equalityComparer.Equals(SizeAsSource, other.SizeAsSource)
            && _equalityComparer.Equals(LeftAsTarget, other.LeftAsTarget)
            && _equalityComparer.Equals(RightAsTarget, other.RightAsTarget)
            && _equalityComparer.Equals(SizeAsTarget, other.SizeAsTarget);

        /// <summary>
        /// <para>
        /// Gets the hash code.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The int</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override int GetHashCode() => (Source, Target, LeftAsSource, RightAsSource,
        ↪  SizeAsSource, LeftAsTarget, RightAsTarget, SizeAsTarget).GetHashCode();

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

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool operator !=(RawLink<TLink> left, RawLink<TLink> right) => !(left ==
        ↪  right);
    }
}
```

1.90 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMetho

```
using System.Runtime.CompilerServices;
using Platform.Data.Doublets.Memory.United.Generic;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 links recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{uint}"/>
    public unsafe abstract class UInt32LinksRecursionlessSizeBalancedTreeMethodsBase :
    ↪  LinksRecursionlessSizeBalancedTreeMethodsBase<uint>
    {
```

```
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLink<uint>* Links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly LinksHeader<uint>* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
        ↪  instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        protected UInt32LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<uint>
        ↪  constants, RawLink<uint>* links, LinksHeader<uint>* header)
            : base(constants, (byte*)links, (byte*)header)
        {
            Links = links;
            Header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetZero() => 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(uint value) => value == 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(uint first, uint second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(uint value) => value > 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(uint first, uint second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than zero.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThanZero(uint value) => true; // value >= 0 is
        ↪  always true for uint

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThanZero(uint value) => value == 0U; // value is
        ↪  always >= 0 for uint

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(uint first, uint second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThanZero(uint value) => false; // value < 0 is always false
        ↪  for uint

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
```

```
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(uint first, uint second) => first < second;

        /// <summary>
        /// <para>
        /// Increments the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Increment(uint value) => ++value;

        /// <summary>
        /// <para>
        /// Decrements the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Decrement(uint value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Add(uint first, uint second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
```

```
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Subtract(uint first, uint second) => first - second;

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheLeftOfSecond(uint first, uint second)
        {
            ref var firstLink = ref Links[first];
            ref var secondLink = ref Links[second];
            return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
             ↪  secondLink.Source, secondLink.Target);
        }

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the right of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheRightOfSecond(uint first, uint second)
        {
            ref var firstLink = ref Links[first];
            ref var secondLink = ref Links[second];
            return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
             ↪  secondLink.Source, secondLink.Target);
        }

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<uint> GetHeaderReference() => ref *Header;
```

```

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLink<uint> GetLinkReference(uint link) => ref Links[link];
    }
}
```

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

```
using System.Runtime.CompilerServices;
using Platform.Data.Doublets.Memory.United.Generic;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 links size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksSizeBalancedTreeMethodsBase{uint}"/>
    public unsafe abstract class UInt32LinksSizeBalancedTreeMethodsBase :
    ↪ LinksSizeBalancedTreeMethodsBase<uint>
    {
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLink<uint>* Links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly LinksHeader<uint>* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32LinksSizeBalancedTreeMethodsBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        protected UInt32LinksSizeBalancedTreeMethodsBase(LinksConstants<uint> constants,
        ↪ RawLink<uint>* links, LinksHeader<uint>* header)
            : base(constants, (byte*)links, (byte*)header)
        {
            Links = links;
            Header = header;
        }

```

```
        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetZero() => 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(uint value) => value == 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(uint first, uint second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(uint value) => value > 0U;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </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>
```

```
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Decrement(uint value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Add(uint first, uint second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Subtract(uint first, uint second) => first - second;

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheLeftOfSecond(uint first, uint second)
        {
            ref var firstLink = ref Links[first];
            ref var secondLink = ref Links[second];
            return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
             ↪  secondLink.Source, secondLink.Target);
        }

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

1.92 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksSourcesRecursionlessSizeBalancedTre

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

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

```
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetRight(uint node) => Links[node].RightAsSource;

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(uint node, uint left) => Links[node].LeftAsSource = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(uint node, uint right) => Links[node].RightAsSource =
        ↪  right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetSize(uint node) => Links[node].SizeAsSource;

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(uint node, uint size) => Links[node].SizeAsSource = size;

```

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

```
        /// <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(uint firstSource, uint firstTarget,
        ↪  uint secondSource, uint secondTarget)
            => firstSource > secondSource || (firstSource == secondSource && firstTarget >
            ↪  secondTarget);

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(uint node)
        {
            ref var link = ref Links[node];
            link.LeftAsSource = 0U;
            link.RightAsSource = 0U;
            link.SizeAsSource = 0U;
        }
    }
}
```

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 links sources size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32LinksSourcesSizeBalancedTreeMethods :
    ↪  UInt32LinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32LinksSourcesSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        public UInt32LinksSourcesSizeBalancedTreeMethods(LinksConstants<uint> constants,
        ↪  RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }

        /// <summary>
        /// <para>
        /// Gets the left reference using the specified node.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref uint GetLeftReference(uint node) => ref Links[node].LeftAsSource;

        /// <summary>
        /// <para>
        /// Gets the right reference using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref uint GetRightReference(uint node) => ref
        ↪  Links[node].RightAsSource;

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

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
```

```
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(uint node, uint left) => Links[node].LeftAsSource = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(uint node, uint right) => Links[node].RightAsSource =
        ↪  right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetSize(uint node) => Links[node].SizeAsSource;

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(uint node, uint size) => Links[node].SizeAsSource = size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint 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 uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetBasePartValue(uint 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(uint firstSource, uint firstTarget,
        ↪  uint secondSource, uint 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(uint firstSource, uint firstTarget,
        ↪  uint secondSource, uint secondTarget)
            => firstSource > secondSource || (firstSource == secondSource && firstTarget >
            ↪  secondTarget);

        /// <summary>
        /// <para>
        /// Clears the node using the specified node.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(uint node)
        {
            ref var link = ref Links[node];
            link.LeftAsSource = 0U;
            link.RightAsSource = 0U;
            link.SizeAsSource = 0U;
        }
    }
}
```

1.94 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTre

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 links targets recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        public UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
        ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
        ↪ header) { }

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

        /// <summary>
        /// <para>
        /// Gets the right reference using the specified node.
        /// </para>
```

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

```
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(uint node, uint right) => Links[node].RightAsTarget =
        ↪  right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetSize(uint node) => Links[node].SizeAsTarget;

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(uint node, uint size) => Links[node].SizeAsTarget = size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetTreeRoot() => Header->RootAsTarget;

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

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="firstSource">
        /// <para>The first source.</para>
        /// <para></para>
```

```
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,
              ↪ uint secondSource, uint secondTarget)
230             => firstTarget < secondTarget || (firstTarget == secondTarget && 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(uint firstSource, uint firstTarget,
              ↪ uint secondSource, uint secondTarget)
260             => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
                  ↪ secondSource);
261 
262         /// <summary>
263         /// <para>
264         /// Clears the node using the specified node.
265         /// </para>
266         /// <para></para>
267         /// </summary>
268         /// <param name="node">
269         /// <para>The node.</para>
270         /// <para></para>
271         /// </param>
272         [MethodImpl(MethodImplOptions.AggressiveInlining)]
273         protected override void ClearNode(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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 links targets size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32LinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32LinksTargetsSizeBalancedTreeMethods :
    ↪ UInt32LinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32LinksTargetsSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        public UInt32LinksTargetsSizeBalancedTreeMethods(LinksConstants<uint> constants,
        ↪ RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links, header) { }

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

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

        /// <summary>
        /// <para>
        /// Gets the left using the specified node.
        /// </para>
        /// <para></para>
```

```
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetLeft(uint node) => Links[node].LeftAsTarget;

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(uint node, uint left) => Links[node].LeftAsTarget = left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(uint node, uint right) => Links[node].RightAsTarget =
        ↪  right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
```

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetSize(uint node) => Links[node].SizeAsTarget;

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(uint node, uint size) => Links[node].SizeAsTarget = size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetTreeRoot() => Header->RootAsTarget;

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

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

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

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

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

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

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

```
        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link index.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLink<uint> GetLinkReference(uint linkIndex) => ref
        ↪  _links[linkIndex];

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(uint first, uint second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(uint first, uint second) => first < second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
```

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(uint first, uint second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(uint first, uint second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(uint first, uint second) => first >= second;

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetZero() => 0U;

        /// <summary>
        /// <para>
        /// Gets the one.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint GetOne() => 1U;

        /// <summary>
        /// <para>
        /// Converts the to int 64 using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
```

```
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The long</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override long ConvertToInt64(uint value) => (long)value;

        /// <summary>
        /// <para>
        /// Converts the to address using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint ConvertToAddress(long value) => (uint)value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Add(uint first, uint second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Subtract(uint first, uint second) => first - second;

        /// <summary>
        /// <para>
        /// Increments the link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
```

```
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Increment(uint link) => ++link;

        /// <summary>
        /// <para>
        /// Decrements the link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override uint Decrement(uint link) => --link;
    }
}
```

1.97 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs

```
using System.Runtime.CompilerServices;
using Platform.Data.Doublets.Memory.United.Generic;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 32 unused links list methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UnusedLinksListMethods{uint}"/>
    public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<uint>
    {
        private readonly RawLink<uint>* _links;
        private readonly LinksHeader<uint>* _header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32UnusedLinksListMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt32UnusedLinksListMethods(RawLink<uint>* links, LinksHeader<uint>* header)
            : base((byte*)links, (byte*)header)
        {
            _links = links;
            _header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
```

```
        /// <returns>
        /// <para>A ref raw link of uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLink<uint> GetLinkReference(uint link) => ref _links[link];

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of uint</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
    }
}
```

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

```
using System.Runtime.CompilerServices;
using Platform.Data.Doublets.Memory.United.Generic;
using static System.Runtime.CompilerServices.Unsafe;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 links avl balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksAvlBalancedTreeMethodsBase{ulong}"/>
    public unsafe abstract class UInt64LinksAvlBalancedTreeMethodsBase :
    ↪  LinksAvlBalancedTreeMethodsBase<ulong>
    {
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly RawLink<ulong>* Links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        protected new readonly LinksHeader<ulong>* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64LinksAvlBalancedTreeMethodsBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        protected UInt64LinksAvlBalancedTreeMethodsBase(LinksConstants<ulong> constants,
        ↪  RawLink<ulong>* links, LinksHeader<ulong>* header)
            : base(constants, (byte*)links, (byte*)header)
        {
            Links = links;
```

```
            Header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetZero() => 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(ulong value) => value == 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(ulong first, ulong second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(ulong value) => value > 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
```

```
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(ulong first, ulong second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
        ↪  always true for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
        ↪  always >= 0 for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <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>
```

```
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Decrement(ulong value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Add(ulong first, ulong second) => first + second;

        /// <summary>
        /// <para>
        /// Subtracts the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Subtract(ulong first, ulong second) => first - second;

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheLeftOfSecond(ulong first, ulong second)
        {
            ref var firstLink = ref Links[first];
            ref var secondLink = ref Links[second];
            return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
            ↪  secondLink.Source, secondLink.Target);
        }

```

```
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,
                    ↪  secondLink.Source, secondLink.Target);
387             }
388
389             /// <summary>
390             /// <para>
391             /// Gets the size value using the specified value.
392             /// </para>
393             /// <para></para>
394             /// </summary>
395             /// <param name="value">
396             /// <para>The value.</para>
397             /// <para></para>
398             /// </param>
399             /// <returns>
400             /// <para>The ulong</para>
401             /// <para></para>
402             /// </returns>
403             [MethodImpl(MethodImplOptions.AggressiveInlining)]
404             protected override ulong GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;
405
406             /// <summary>
407             /// <para>
408             /// Sets the size value using the specified stored value.
409             /// </para>
410             /// <para></para>
411             /// </summary>
412             /// <param name="storedValue">
413             /// <para>The stored value.</para>
414             /// <para></para>
415             /// </param>
416             /// <param name="size">
417             /// <para>The size.</para>
418             /// <para></para>
419             /// </param>
420             [MethodImpl(MethodImplOptions.AggressiveInlining)]
421             protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
                ↪  storedValue & 31UL | (size & 134217727UL) << 5;
422
423             /// <summary>
424             /// <para>
425             /// Determines whether this instance get left is child value.
426             /// </para>
427             /// <para></para>
428             /// </summary>
429             /// <param name="value">
430             /// <para>The value.</para>
431             /// <para></para>
432             /// </param>
433             /// <returns>
434             /// <para>The bool</para>
435             /// <para></para>
436             /// </returns>
437             [MethodImpl(MethodImplOptions.AggressiveInlining)]
438             protected override bool GetLeftIsChildValue(ulong value) => (value & 16UL) >> 4 == 1UL;
```

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

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

1.99 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMetho

```
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 :
        ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<ulong>
16      {
17          /// <summary>
18          /// <para>
19          /// The links.
20          /// </para>
21          /// <para></para>
22          /// </summary>
23          protected new readonly RawLink<ulong>* Links;
24          /// <summary>
25          /// <para>
26          /// The header.
27          /// </para>
28          /// <para></para>
29          /// </summary>
30          protected new readonly LinksHeader<ulong>* Header;
```

```
31
32          /// <summary>
33          /// <para>
34          /// Initializes a new <see cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
            ↪  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 UInt64LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<ulong>
            ↪  constants, RawLink<ulong>* links, LinksHeader<ulong>* header)
51              : base(constants, (byte*)links, (byte*)header)
52          {
53              Links = links;
54              Header = header;
55          }
56
57          /// <summary>
58          /// <para>
59          /// Gets the zero.
60          /// </para>
61          /// <para></para>
62          /// </summary>
63          /// <returns>
64          /// <para>The ulong</para>
65          /// <para></para>
66          /// </returns>
67          [MethodImpl(MethodImplOptions.AggressiveInlining)]
68          protected override ulong GetZero() => 0UL;
69
70          /// <summary>
71          /// <para>
72          /// Determines whether this instance equal to zero.
73          /// </para>
74          /// <para></para>
75          /// </summary>
76          /// <param name="value">
77          /// <para>The value.</para>
78          /// <para></para>
79          /// </param>
80          /// <returns>
81          /// <para>The bool</para>
82          /// <para></para>
83          /// </returns>
84          [MethodImpl(MethodImplOptions.AggressiveInlining)]
85          protected override bool EqualToZero(ulong value) => value == 0UL;
86
87          /// <summary>
88          /// <para>
89          /// Determines whether this instance are equal.
90          /// </para>
91          /// <para></para>
92          /// </summary>
93          /// <param name="first">
94          /// <para>The first.</para>
95          /// <para></para>
96          /// </param>
97          /// <param name="second">
98          /// <para>The second.</para>
99          /// <para></para>
100         /// </param>
101         /// <returns>
102         /// <para>The bool</para>
103         /// <para></para>
104         /// </returns>
105         [MethodImpl(MethodImplOptions.AggressiveInlining)]
106         protected override bool AreEqual(ulong first, ulong second) => first == second;
```

```
107
108         /// <summary>
109         /// <para>
110         /// Determines whether this instance greater than zero.
111         /// </para>
112         /// <para></para>
113         /// </summary>
114         /// <param name="value">
115         /// <para>The value.</para>
116         /// <para></para>
117         /// </param>
118         /// <returns>
119         /// <para>The bool</para>
120         /// <para></para>
121         /// </returns>
122         [MethodImpl(MethodImplOptions.AggressiveInlining)]
123         protected override bool GreaterThanZero(ulong value) => value > 0UL;
124
125         /// <summary>
126         /// <para>
127         /// Determines whether this instance greater than.
128         /// </para>
129         /// <para></para>
130         /// </summary>
131         /// <param name="first">
132         /// <para>The first.</para>
133         /// <para></para>
134         /// </param>
135         /// <param name="second">
136         /// <para>The second.</para>
137         /// <para></para>
138         /// </param>
139         /// <returns>
140         /// <para>The bool</para>
141         /// <para></para>
142         /// </returns>
143         [MethodImpl(MethodImplOptions.AggressiveInlining)]
144         protected override bool GreaterThan(ulong first, ulong second) => first > second;
145
146         /// <summary>
147         /// <para>
148         /// Determines whether this instance greater or equal than.
149         /// </para>
150         /// <para></para>
151         /// </summary>
152         /// <param name="first">
153         /// <para>The first.</para>
154         /// <para></para>
155         /// </param>
156         /// <param name="second">
157         /// <para>The second.</para>
158         /// <para></para>
159         /// </param>
160         /// <returns>
161         /// <para>The bool</para>
162         /// <para></para>
163         /// </returns>
164         [MethodImpl(MethodImplOptions.AggressiveInlining)]
165         protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
166
167         /// <summary>
168         /// <para>
169         /// Determines whether this instance greater or equal than zero.
170         /// </para>
171         /// <para></para>
172         /// </summary>
173         /// <param name="value">
174         /// <para>The value.</para>
175         /// <para></para>
176         /// </param>
177         /// <returns>
178         /// <para>The bool</para>
179         /// <para></para>
180         /// </returns>
181         [MethodImpl(MethodImplOptions.AggressiveInlining)]
182         protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
            ↪  always true for ulong
183
```

```
        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
        ↪  always >= 0 for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
        ↪  for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(ulong first, ulong second) => first < second;

```

```
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>
```

```
        /// Determines whether this instance first is to the left of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheLeftOfSecond(ulong first, ulong second)
        {
            ref var firstLink = ref Links[first];
            ref var secondLink = ref Links[second];
            return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
             ↪  secondLink.Source, secondLink.Target);
        }

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the right of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheRightOfSecond(ulong first, ulong second)
        {
            ref var firstLink = ref Links[first];
            ref var secondLink = ref Links[second];
            return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
             ↪  secondLink.Source, secondLink.Target);
        }

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<ulong> GetHeaderReference() => ref *Header;

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of ulong</para>
        /// <para></para>
```

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

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

```
1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 links size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksSizeBalancedTreeMethodsBase{ulong}"/>
15     public unsafe abstract class UInt64LinksSizeBalancedTreeMethodsBase :
       ↪  LinksSizeBalancedTreeMethodsBase<ulong>
16     {
17         /// <summary>
18         /// <para>
19         /// The links.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         protected new readonly RawLink<ulong>* Links;
24         /// <summary>
25         /// <para>
26         /// The header.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         protected new readonly LinksHeader<ulong>* Header;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="UInt64LinksSizeBalancedTreeMethodsBase"/> instance.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="constants">
39         /// <para>A constants.</para>
40         /// <para></para>
41         /// </param>
42         /// <param name="links">
43         /// <para>A links.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="header">
47         /// <para>A header.</para>
48         /// <para></para>
49         /// </param>
50         protected UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
           ↪  RawLink<ulong>* links, LinksHeader<ulong>* header)
51             : base(constants, (byte*)links, (byte*)header)
52         {
53             Links = links;
54             Header = header;
55         }
56
57         /// <summary>
58         /// <para>
59         /// Gets the zero.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <returns>
64         /// <para>The ulong</para>
65         /// <para></para>
66         /// </returns>
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         protected override ulong GetZero() => 0UL;
69
70         /// <summary>
```

```
        /// <para>
        /// Determines whether this instance equal to zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool EqualToZero(ulong value) => value == 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(ulong first, ulong second) => first == second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThanZero(ulong value) => value > 0UL;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(ulong first, ulong second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThanZero(ulong value) => true; // value >= 0 is
        ↪  always true for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
        ↪  always >= 0 for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less than zero.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThanZero(ulong value) => false; // value < 0 is always false
        ↪  for ulong

        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(ulong first, ulong second) => first < second;

        /// <summary>
        /// <para>
        /// Increments the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Increment(ulong value) => ++value;

        /// <summary>
        /// <para>
        /// Decrements the value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong Decrement(ulong value) => --value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
```

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

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

1.101 ./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 :
       ↪  UInt64LinksAvlBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="UInt64LinksSourcesAvlBalancedTreeMethods"/> 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 UInt64LinksSourcesAvlBalancedTreeMethods(LinksConstants<ulong> constants,
            ↪  RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
            ↪  { }
35
36          /// <summary>
37          /// <para>
38          /// Gets the left reference using the specified node.
39          /// </para>
40          /// <para></para>
41          /// </summary>
42          /// <param name="node">
43          /// <para>The node.</para>
44          /// <para></para>
45          /// </param>
46          /// <returns>
47          /// <para>The ref ulong</para>
48          /// <para></para>
49          /// </returns>
50          [MethodImpl(MethodImplOptions.AggressiveInlining)]
51          protected override ref ulong GetLeftReference(ulong node) => ref
            ↪  Links[node].LeftAsSource;
52
53          /// <summary>
54          /// <para>
55          /// Gets the right reference using the specified node.
56          /// </para>
57          /// <para></para>
58          /// </summary>
59          /// <param name="node">
60          /// <para>The node.</para>
61          /// <para></para>
62          /// </param>
63          /// <returns>
64          /// <para>The ref ulong</para>
65          /// <para></para>
66          /// </returns>
67          [MethodImpl(MethodImplOptions.AggressiveInlining)]
68          protected override ref ulong GetRightReference(ulong node) => ref
            ↪  Links[node].RightAsSource;
69
70          /// <summary>
71          /// <para>
72          /// Gets the left using the specified node.
73          /// </para>
74          /// <para></para>
75          /// </summary>
76          /// <param name="node">
77          /// <para>The node.</para>
78          /// <para></para>
79          /// </param>
80          /// <returns>
81          /// <para>The ulong</para>
82          /// <para></para>
83          /// </returns>
84          [MethodImpl(MethodImplOptions.AggressiveInlining)]
85          protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
86
87          /// <summary>
88          /// <para>
89          /// Gets the right using the specified node.
90          /// </para>
91          /// <para></para>
92          /// </summary>
93          /// <param name="node">
94          /// <para>The node.</para>
95          /// <para></para>
96          /// </param>
97          /// <returns>
98          /// <para>The ulong</para>
99          /// <para></para>
100         /// </returns>
101         [MethodImpl(MethodImplOptions.AggressiveInlining)]
102         protected override ulong GetRight(ulong node) => Links[node].RightAsSource;
103
104         /// <summary>
105         /// <para>
106         /// Sets the left using the specified node.
107         /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsSource =
        ↪  left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
        ↪  right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetSize(ulong node) => GetSizeValue(Links[node].SizeAsSource);

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(ulong node, ulong size) => SetSizeValue(ref
        ↪  Links[node].SizeAsSource, size);

        /// <summary>
        /// <para>
        /// Determines whether this instance get left is child.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <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) =>
            ↪ GetRightIsChildValue(Links[node].SizeAsSource);
225
226         //[MethodImpl(MethodImplOptions.AggressiveInlining)]
227         //protected override bool GetRightIsChild(ulong node) => IsChild(node, GetRight(node));
228
229         /// <summary>
230         /// <para>
231         /// Sets the right is child 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         /// <param name="value">
240         /// <para>The value.</para>
241         /// <para></para>
242         /// </param>
243         [MethodImpl(MethodImplOptions.AggressiveInlining)]
244         protected override void SetRightIsChild(ulong node, bool value) =>
            ↪ SetRightIsChildValue(ref Links[node].SizeAsSource, value);
245
246         /// <summary>
247         /// <para>
248         /// Gets the balance using the specified node.
249         /// </para>
250         /// <para></para>
251         /// </summary>
252         /// <param name="node">
253         /// <para>The node.</para>
254         /// <para></para>
255         /// </param>
256         /// <returns>
```

```
        /// <para>The sbyte</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override sbyte GetBalance(ulong node) =>
        ↪  GetBalanceValue(Links[node].SizeAsSource);

        /// <summary>
        /// <para>
        /// Sets the balance using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
        ↪  Links[node].SizeAsSource, value);

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </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>
```

```
333         /// <para>The bool</para>
334         /// <para></para>
335         /// </returns>
336         [MethodImpl(MethodImplOptions.AggressiveInlining)]
337         protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
            ↪ ulong secondSource, ulong secondTarget)
338             => firstSource < secondSource || (firstSource == secondSource && firstTarget <
                ↪ secondTarget);
339 
340         /// <summary>
341         /// <para>
342         /// Determines whether this instance first is to the right of second.
343         /// </para>
344         /// <para></para>
345         /// </summary>
346         /// <param name="firstSource">
347         /// <para>The first source.</para>
348         /// <para></para>
349         /// </param>
350         /// <param name="firstTarget">
351         /// <para>The first target.</para>
352         /// <para></para>
353         /// </param>
354         /// <param name="secondSource">
355         /// <para>The second source.</para>
356         /// <para></para>
357         /// </param>
358         /// <param name="secondTarget">
359         /// <para>The second target.</para>
360         /// <para></para>
361         /// </param>
362         /// <returns>
363         /// <para>The bool</para>
364         /// <para></para>
365         /// </returns>
366         [MethodImpl(MethodImplOptions.AggressiveInlining)]
367         protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
            ↪ ulong secondSource, ulong secondTarget)
368             => firstSource > secondSource || (firstSource == secondSource && firstTarget >
                ↪ secondTarget);
369 
370         /// <summary>
371         /// <para>
372         /// Clears the node using the specified node.
373         /// </para>
374         /// <para></para>
375         /// </summary>
376         /// <param name="node">
377         /// <para>The node.</para>
378         /// <para></para>
379         /// </param>
380         [MethodImpl(MethodImplOptions.AggressiveInlining)]
381         protected override void ClearNode(ulong node)
382         {
383             ref var link = ref Links[node];
384             link.LeftAsSource = 0UL;
385             link.RightAsSource = 0UL;
386             link.SizeAsSource = 0UL;
387         }
388     }
389 }
```

1.102 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTr

```
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 :
        ↪ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
```

```
15      {
16          /// <summary>
17          /// <para>
18          /// Initializes a new <see
            ↪  cref="UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(LinksConstants<ulong>
            ↪  constants, RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants,
            ↪  links, header) { }
35
36          /// <summary>
37          /// <para>
38          /// Gets the left reference using the specified node.
39          /// </para>
40          /// <para></para>
41          /// </summary>
42          /// <param name="node">
43          /// <para>The node.</para>
44          /// <para></para>
45          /// </param>
46          /// <returns>
47          /// <para>The ref ulong</para>
48          /// <para></para>
49          /// </returns>
50          [MethodImpl(MethodImplOptions.AggressiveInlining)]
51          protected override ref ulong GetLeftReference(ulong node) => ref
            ↪  Links[node].LeftAsSource;
52
53          /// <summary>
54          /// <para>
55          /// Gets the right reference using the specified node.
56          /// </para>
57          /// <para></para>
58          /// </summary>
59          /// <param name="node">
60          /// <para>The node.</para>
61          /// <para></para>
62          /// </param>
63          /// <returns>
64          /// <para>The ref ulong</para>
65          /// <para></para>
66          /// </returns>
67          [MethodImpl(MethodImplOptions.AggressiveInlining)]
68          protected override ref ulong GetRightReference(ulong node) => ref
            ↪  Links[node].RightAsSource;
69
70          /// <summary>
71          /// <para>
72          /// Gets the left using the specified node.
73          /// </para>
74          /// <para></para>
75          /// </summary>
76          /// <param name="node">
77          /// <para>The node.</para>
78          /// <para></para>
79          /// </param>
80          /// <returns>
81          /// <para>The ulong</para>
82          /// <para></para>
83          /// </returns>
84          [MethodImpl(MethodImplOptions.AggressiveInlining)]
85          protected override ulong GetLeft(ulong node) => Links[node].LeftAsSource;
86
```

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

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

```
        /// <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>
        /// Clears the node using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ClearNode(ulong node)
        {
            ref var link = ref Links[node];
            link.LeftAsSource = 0UL;
            link.RightAsSource = 0UL;
            link.SizeAsSource = 0UL;
        }
    }
}
```

1.103 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesSizeBalancedTreeMethods.c

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 links sources size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64LinksSourcesSizeBalancedTreeMethods :
    ↪  UInt64LinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64LinksSourcesSizeBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
```

```
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        public UInt64LinksSourcesSizeBalancedTreeMethods(LinksConstants<ulong> constants,
        ↪  RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
        ↪  { }

        /// <summary>
        /// <para>
        /// Gets the left reference using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref ulong GetLeftReference(ulong node) => ref
        ↪  Links[node].LeftAsSource;

        /// <summary>
        /// <para>
        /// Gets the right reference using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ref ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref ulong GetRightReference(ulong node) => ref
        ↪  Links[node].RightAsSource;

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

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

```

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

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

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

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

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

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

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

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 links targets avl balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
    public unsafe class UInt64LinksTargetsAvlBalancedTreeMethods :
    ↪  UInt64LinksAvlBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64LinksTargetsAvlBalancedTreeMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="constants">
        /// <para>A constants.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        public UInt64LinksTargetsAvlBalancedTreeMethods(LinksConstants<ulong> constants,
        ↪  RawLink<ulong>* links, LinksHeader<ulong>* header) : base(constants, links, header)
        ↪  { }

        /// <summary>
        /// <para>
        /// Gets the left reference using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
```

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

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

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

```
269         /// <para></para>
270         /// </param>
271         [MethodImpl(MethodImplOptions.AggressiveInlining)]
272         protected override void SetBalance(ulong node, sbyte value) => SetBalanceValue(ref
            ↪ Links[node].SizeAsTarget, value);
273
274         /// <summary>
275         /// <para>
276         /// Gets the tree root.
277         /// </para>
278         /// <para></para>
279         /// </summary>
280         /// <returns>
281         /// <para>The ulong</para>
282         /// <para></para>
283         /// </returns>
284         [MethodImpl(MethodImplOptions.AggressiveInlining)]
285         protected override ulong GetTreeRoot() => Header->RootAsTarget;
286
287         /// <summary>
288         /// <para>
289         /// Gets the base part value using the specified link.
290         /// </para>
291         /// <para></para>
292         /// </summary>
293         /// <param name="link">
294         /// <para>The link.</para>
295         /// <para></para>
296         /// </param>
297         /// <returns>
298         /// <para>The ulong</para>
299         /// <para></para>
300         /// </returns>
301         [MethodImpl(MethodImplOptions.AggressiveInlining)]
302         protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
303
304         /// <summary>
305         /// <para>
306         /// Determines whether this instance first is to the left of second.
307         /// </para>
308         /// <para></para>
309         /// </summary>
310         /// <param name="firstSource">
311         /// <para>The first source.</para>
312         /// <para></para>
313         /// </param>
314         /// <param name="firstTarget">
315         /// <para>The first target.</para>
316         /// <para></para>
317         /// </param>
318         /// <param name="secondSource">
319         /// <para>The second source.</para>
320         /// <para></para>
321         /// </param>
322         /// <param name="secondTarget">
323         /// <para>The second target.</para>
324         /// <para></para>
325         /// </param>
326         /// <returns>
327         /// <para>The bool</para>
328         /// <para></para>
329         /// </returns>
330         [MethodImpl(MethodImplOptions.AggressiveInlining)]
331         protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
            ↪ ulong secondSource, ulong secondTarget)
332             => firstTarget < secondTarget || (firstTarget == secondTarget && firstSource <
                ↪ secondSource);
333
334         /// <summary>
335         /// <para>
336         /// Determines whether this instance first is to the right of second.
337         /// </para>
338         /// <para></para>
339         /// </summary>
340         /// <param name="firstSource">
341         /// <para>The first source.</para>
342         /// <para></para>
343         /// </param>
```

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

1.105 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTr

```
1  using System.Runtime.CompilerServices;
2  
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4  
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links targets recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods :
        ↪ UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see
            ↪ cref="UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="constants">
23         /// <para>A constants.</para>
24         /// <para></para>
25         /// </param>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>
32         /// <para></para>
```

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

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsTarget =
        ↪  left;

        /// <summary>
        /// <para>
        /// Sets the right using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="right">
        /// <para>The right.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetRight(ulong node, ulong right) => Links[node].RightAsTarget =
        ↪  right;

        /// <summary>
        /// <para>
        /// Gets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetSize(ulong node) => Links[node].SizeAsTarget;

        /// <summary>
        /// <para>
        /// Sets the size using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSize(ulong node, ulong size) => Links[node].SizeAsTarget =
        ↪  size;

        /// <summary>
        /// <para>
        /// Gets the tree root.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
```

```
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetTreeRoot() => Header->RootAsTarget;

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

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

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

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

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

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

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

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

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

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

        /// <summary>
        /// <para>
        /// Sets the left using the specified node.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <param name="left">
        /// <para>The left.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetLeft(ulong node, ulong left) => Links[node].LeftAsTarget =
        ↪ left;

        /// <summary>
        /// <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,
                ↪  ulong secondSource, ulong secondTarget)
230                 => firstTarget < secondTarget || (firstTarget == secondTarget && 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(ulong firstSource, ulong firstTarget,
                ↪  ulong secondSource, ulong secondTarget)
260                 => firstTarget > secondTarget || (firstTarget == secondTarget && firstSource >
                    ↪  secondSource);
261
262             /// <summary>
263             /// <para>
264             /// Clears the node using the specified node.
265             /// </para>
266             /// <para></para>
267             /// </summary>
268             /// <param name="node">
269             /// <para>The node.</para>
270             /// <para></para>
271             /// </param>
```

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

1.107 ./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
        ↪  organizing the storage of links with addresses represented as <see cref="ulong"
        ↪  />.</para>
13      /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным
        ↪  размером, для организации хранения связей с адресами представленными в виде <see
        ↪  cref="ulong"/>.</para>
14      /// </summary>
15      public unsafe class UInt64UnitedMemoryLinks : UnitedMemoryLinksBase<ulong>
16      {
17          private readonly Func<ILinksTreeMethods<ulong>> _createSourceTreeMethods;
18          private readonly Func<ILinksTreeMethods<ulong>> _createTargetTreeMethods;
19          private LinksHeader<ulong>* _header;
20          private RawLink<ulong>* _links;
21
22          /// <summary>
23          /// <para>
24          /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
25          /// </para>
26          /// <para></para>
27          /// </summary>
28          /// <param name="address">
29          /// <para>A address.</para>
30          /// <para></para>
31          /// </param>
32          [MethodImpl(MethodImplOptions.AggressiveInlining)]
33          public UInt64UnitedMemoryLinks(string address) : this(address, DefaultLinksSizeStep) { }
34
35          /// <summary>
36          /// Создаёт экземпляр базы данных Links в файле по указанному адресу, с указанным
            ↪  минимальным шагом расширения базы данных.
37          /// </summary>
38          /// <param name="address">Полный пусть к файлу базы данных.</param>
39          /// <param name="memoryReservationStep">Минимальный шаг расширения базы данных в
            ↪  байтах.</param>
40          [MethodImpl(MethodImplOptions.AggressiveInlining)]
41          public UInt64UnitedMemoryLinks(string address, long memoryReservationStep) : this(new
            ↪  FileMappedResizableDirectMemory(address, memoryReservationStep),
            ↪  memoryReservationStep) { }
42
43          /// <summary>
44          /// <para>
45          /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
46          /// </para>
47          /// <para></para>
48          /// </summary>
49          /// <param name="memory">
50          /// <para>A memory.</para>
51          /// <para></para>
52          /// </param>
53          [MethodImpl(MethodImplOptions.AggressiveInlining)]
54          public UInt64UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
            ↪  DefaultLinksSizeStep) { }
55
56          /// <summary>
57          /// <para>
58          /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
```

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

```
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetPointers(IResizableDirectMemory memory)
        {
            _header = (LinksHeader<ulong>*)memory.Pointer;
            _links = (RawLink<ulong>*)memory.Pointer;
            SourcesTreeMethods = _createSourceTreeMethods();
            TargetsTreeMethods = _createTargetTreeMethods();
            UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
        }

        /// <summary>
        /// <para>
        /// Resets the pointers.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ResetPointers()
        {
            base.ResetPointers();
            _links = null;
            _header = null;
        }

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link index.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
        ↪  _links[linkIndex];

        /// <summary>
        /// <para>
        /// Determines whether this instance are equal.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool AreEqual(ulong first, ulong second) => first == second;

```

```
        /// <summary>
        /// <para>
        /// Determines whether this instance less than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessThan(ulong first, ulong second) => first < second;

        /// <summary>
        /// <para>
        /// Determines whether this instance less or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool LessOrEqualThan(ulong first, ulong second) => first <= second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterThan(ulong first, ulong second) => first > second;

        /// <summary>
        /// <para>
        /// Determines whether this instance greater or equal than.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

```
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;

        /// <summary>
        /// <para>
        /// Gets the zero.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetZero() => 0UL;

        /// <summary>
        /// <para>
        /// Gets the one.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong GetOne() => 1UL;

        /// <summary>
        /// <para>
        /// Converts the to int 64 using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The long</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override long ConvertToInt64(ulong value) => (long)value;

        /// <summary>
        /// <para>
        /// Converts the to address using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ulong ConvertToAddress(long value) => (ulong)value;

        /// <summary>
        /// <para>
        /// Adds the first.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="first">
        /// <para>The first.</para>
        /// <para></para>
        /// </param>
        /// <param name="second">
        /// <para>The second.</para>
```

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

1.108 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs

```
1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3 
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5 
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 unused links list methods.
11     /// </para>
```

```
    /// <para></para>
    /// </summary>
    /// <seealso cref="UnusedLinksListMethods{ulong}"/>
    public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<ulong>
    {
        private readonly RawLink<ulong>* _links;
        private readonly LinksHeader<ulong>* _header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64UnusedLinksListMethods"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt64UnusedLinksListMethods(RawLink<ulong>* links, LinksHeader<ulong>* header)
            : base((byte*)links, (byte*)header)
        {
            _links = links;
            _header = header;
        }

        /// <summary>
        /// <para>
        /// Gets the link reference using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A ref raw link of ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref _links[link];

        /// <summary>
        /// <para>
        /// Gets the header reference.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A ref links header of ulong</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Interfaces;

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

namespace Platform.Data.Doublets.PropertyOperators
{
    /// <summary>
    /// <para>
    /// Represents the properties operator.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksOperatorBase{TLink}"/>
    /// <seealso cref="IProperties{TLink, TLink, TLink}"/>
```

```
17      public class PropertiesOperator<TLink> : LinksOperatorBase<TLink>, IProperties<TLink, TLink,
        ↪  TLink>
18      {
19          private static readonly EqualityComparer<TLink> _equalityComparer =
            ↪  EqualityComparer<TLink>.Default;
20
21          /// <summary>
22          /// <para>
23          /// Initializes a new <see cref="PropertiesOperator"/> instance.
24          /// </para>
25          /// <para></para>
26          /// </summary>
27          /// <param name="links">
28          /// <para>A links.</para>
29          /// <para></para>
30          /// </param>
31          [MethodImpl(MethodImplOptions.AggressiveInlining)]
32          public PropertiesOperator(ILinks<TLink> links) : base(links) { }
33
34          /// <summary>
35          /// <para>
36          /// Gets the value using the specified object.
37          /// </para>
38          /// <para></para>
39          /// </summary>
40          /// <param name="@object">
41          /// <para>The object.</para>
42          /// <para></para>
43          /// </param>
44          /// <param name="property">
45          /// <para>The property.</para>
46          /// <para></para>
47          /// </param>
48          /// <returns>
49          /// <para>The link</para>
50          /// <para></para>
51          /// </returns>
52          [MethodImpl(MethodImplOptions.AggressiveInlining)]
53          public TLink GetValue(TLink @object, TLink property)
54          {
55              var links = _links;
56              var objectProperty = links.SearchOrDefault(@object, property);
57              if (_equalityComparer.Equals(objectProperty, default))
58              {
59                  return default;
60              }
61              var constants = links.Constants;
62              var any = constants.Any;
63              var query = new Link<TLink>(any, objectProperty, any);
64              var valueLink = links.SingleOrDefault(query);
65              if (valueLink == null)
66              {
67                  return default;
68              }
69              return links.GetTarget(valueLink[constants.IndexPart]);
70          }
71
72          /// <summary>
73          /// <para>
74          /// Sets the value using the specified object.
75          /// </para>
76          /// <para></para>
77          /// </summary>
78          /// <param name="@object">
79          /// <para>The object.</para>
80          /// <para></para>
81          /// </param>
82          /// <param name="property">
83          /// <para>The property.</para>
84          /// <para></para>
85          /// </param>
86          /// <param name="value">
87          /// <para>The value.</para>
88          /// <para></para>
89          /// </param>
90          [MethodImpl(MethodImplOptions.AggressiveInlining)]
91          public void SetValue(TLink @object, TLink property, TLink value)
92          {
```

```
            var links = _links;
            var objectProperty = links.GetOrCreate(@object, property);
            links.DeleteMany(links.AllIndices(links.Constants.Any, objectProperty));
            links.GetOrCreate(objectProperty, value);
        }
    }
}
```

1.110 ./csharp/Platform.Data.Doublets/PropertyOperators/PropertyOperator.cs

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Interfaces;

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

namespace Platform.Data.Doublets.PropertyOperators
{
    /// <summary>
    /// <para>
    /// Represents the property operator.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksOperatorBase{TLink}"/>
    /// <seealso cref="IProperty{TLink, TLink}"/>
    public class PropertyOperator<TLink> : LinksOperatorBase<TLink>, IProperty<TLink, TLink>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        private readonly TLink _propertyMarker;
        private readonly TLink _propertyValueMarker;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="PropertyOperator"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="propertyMarker">
        /// <para>A property marker.</para>
        /// <para></para>
        /// </param>
        /// <param name="propertyValueMarker">
        /// <para>A property value marker.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public PropertyOperator(ILinks<TLink> links, TLink propertyMarker, TLink
        ↪  propertyValueMarker) : base(links)
        {
            _propertyMarker = propertyMarker;
            _propertyValueMarker = propertyValueMarker;
        }

        /// <summary>
        /// <para>
        /// Gets the link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink Get(TLink link)
        {
            var property = _links.SearchOrDefault(link, _propertyMarker);
            return GetValue(GetContainer(property));
        }
```

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

```
            var links = _links;
            var property = links.GetOrCreate(link, _propertyMarker);
            var container = GetContainer(property);
            if (_equalityComparer.Equals(container, default))
            {
                links.GetOrCreate(property, value);
            }
            else
            {
                links.Update(container, property, value);
            }
        }
    }
}
```

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

```
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Collections.Stacks;

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

namespace Platform.Data.Doublets.Stacks
{
    /// <summary>
    /// <para>
    /// Represents the stack.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksOperatorBase{TLink}"/>
    /// <seealso cref="IStack{TLink}"/>
    public class Stack<TLink> : LinksOperatorBase<TLink>, IStack<TLink>
    {
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        private readonly TLink _stack;

        /// <summary>
        /// <para>
        /// Gets the is empty value.
        /// </para>
        /// <para></para>
        /// </summary>
        public bool IsEmpty
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get => _equalityComparer.Equals(Peek(), _stack);
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Stack"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="stack">
        /// <para>A stack.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Stack(ILinks<TLink> links, TLink stack) : base(links) => _stack = stack;

        /// <summary>
        /// <para>
        /// Gets the stack marker.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

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

1.112 ./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs

```
1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Stacks
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the stack extensions.
```

```
    /// </para>
    /// <para></para>
    /// </summary>
    public static class StackExtensions
    {
        /// <summary>
        /// <para>
        /// Creates the stack using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="stackMarker">
        /// <para>The stack marker.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The stack.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TLink CreateStack<TLink>(this ILinks<TLink> links, TLink stackMarker)
        {
            var stackPoint = links.CreatePoint();
            var stack = links.Update(stackPoint, stackMarker, stackPoint);
            return stack;
        }
    }
}
```

1.113 ./csharp/Platform.Data.Doublets/SynchronizedLinks.cs

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using Platform.Data.Doublets;
using Platform.Threading.Synchronization;

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

namespace Platform.Data.Doublets
{
    /// <remarks>
    /// TODO: Autogeneration of synchronized wrapper (decorator).
    /// TODO: Try to unfold code of each method using IL generation for performance improvements.
    /// TODO: Or even to unfold multiple layers of implementations.
    /// </remarks>
    public class SynchronizedLinks<TLinkAddress> : ISynchronizedLinks<TLinkAddress>
    {
        /// <summary>
        /// <para>
        /// Gets the constants value.
        /// </para>
        /// <para></para>
        /// </summary>
        public LinksConstants<TLinkAddress> Constants
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

        /// <summary>
        /// <para>
        /// Gets the sync root value.
        /// </para>
        /// <para></para>
        /// </summary>
        public ISynchronization SyncRoot
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

```

```
        /// <summary>
        /// <para>
        /// Gets the sync value.
        /// </para>
        /// <para></para>
        /// </summary>
        public ILinks<TLinkAddress> Sync
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

        /// <summary>
        /// <para>
        /// Gets the unsync value.
        /// </para>
        /// <para></para>
        /// </summary>
        public ILinks<TLinkAddress> Unsync
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="SynchronizedLinks"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public SynchronizedLinks(ILinks<TLinkAddress> links) : this(new
        ↪  ReaderWriterLockSynchronization(), links) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="SynchronizedLinks"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="synchronization">
        /// <para>A synchronization.</para>
        /// <para></para>
        /// </param>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public SynchronizedLinks(ISynchronization synchronization, ILinks<TLinkAddress> links)
        {
            SyncRoot = synchronization;
            Sync = this;
            Unsync = links;
            Constants = links.Constants;
        }

        /// <summary>
        /// <para>
        /// Counts the restriction.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restriction">
        /// <para>The restriction.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link address</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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) =>
            ↪ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Create);
156
157         /// <summary>
158         /// <para>
159         /// Updates the restrictions.
160         /// </para>
161         /// <para></para>
162         /// </summary>
163         /// <param name="restrictions">
164         /// <para>The restrictions.</para>
165         /// <para></para>
166         /// </param>
167         /// <param name="substitution">
168         /// <para>The substitution.</para>
169         /// <para></para>
170         /// </param>
171         /// <returns>
172         /// <para>The link address</para>
173         /// <para></para>
174         /// </returns>
175         [MethodImpl(MethodImplOptions.AggressiveInlining)]
176         public TLinkAddress Update(IList<TLinkAddress> restrictions, IList<TLinkAddress>
            ↪ substitution) => SyncRoot.ExecuteWriteOperation(restrictions, substitution,
            ↪ Unsync.Update);
177
178         /// <summary>
179         /// <para>
180         /// Deletes the restrictions.
181         /// </para>
182         /// <para></para>
183         /// </summary>
184         /// <param name="restrictions">
185         /// <para>The restrictions.</para>
186         /// <para></para>
187         /// </param>
188         [MethodImpl(MethodImplOptions.AggressiveInlining)]
189         public void Delete(IList<TLinkAddress> restrictions) =>
            ↪ SyncRoot.ExecuteWriteOperation(restrictions, Unsync.Delete);
```

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

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

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

```
        /// <summary>
        /// <para>
        /// Formats the structure using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <param name="isElement">
        /// <para>The is element.</para>
        /// <para></para>
        /// </param>
        /// <param name="renderIndex">
        /// <para>The render index.</para>
        /// <para></para>
        /// </param>
        /// <param name="renderDebug">
        /// <para>The render debug.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
        ↪  Func<Link<ulong>, bool> isElement, bool renderIndex = false, bool renderDebug =
        ↪  false)
        {
            var sb = new StringBuilder();
            var visited = new HashSet<ulong>();
            links.AppendStructure(sb, visited, linkIndex, isElement, (innerSb, link) =>
            ↪  innerSb.Append(link.Index), renderIndex, renderDebug);
            return sb.ToString();
        }

        /// <summary>
        /// <para>
        /// Formats the structure using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <param name="isElement">
        /// <para>The is element.</para>
        /// <para></para>
        /// </param>
        /// <param name="appendElement">
        /// <para>The append element.</para>
        /// <para></para>
        /// </param>
        /// <param name="renderIndex">
        /// <para>The render index.</para>
        /// <para></para>
        /// </param>
        /// <param name="renderDebug">
        /// <para>The render debug.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

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

```
                        sb.Append(':');
                    }
                    if (link.Source == link.Index)
                    {
                        sb.Append(link.Index);
                    }
                    else
                    {
                        var source = new Link<ulong>(links.GetLink(link.Source));
                        if (isElement(source))
                        {
                            appendElement(sb, source);
                        }
                        else
                        {
                            links.AppendStructure(sb, visited, source.Index, isElement,
                            ↪  appendElement, renderIndex);
                        }
                    }
                    sb.Append(' ');
                    if (link.Target == link.Index)
                    {
                        sb.Append(link.Index);
                    }
                    else
                    {
                        var target = new Link<ulong>(links.GetLink(link.Target));
                        if (isElement(target))
                        {
                            appendElement(sb, target);
                        }
                        else
                        {
                            links.AppendStructure(sb, visited, target.Index, isElement,
                            ↪  appendElement, renderIndex);
                        }
                    }
                    sb.Append(')');
                }
                else
                {
                    if (renderDebug)
                    {
                        sb.Append('*');
                    }
                    sb.Append(linkIndex);
                }
            }
            else
            {
                if (renderDebug)
                {
                    sb.Append('~');
                }
                sb.Append(linkIndex);
            }
        }
    }
}
```

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

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.IO;
using System.Runtime.CompilerServices;
using System.Threading;
using System.Threading.Tasks;
using Platform.Disposables;
using Platform.Timestamps;
using Platform.Unsafe;
using Platform.IO;
using Platform.Data.Doublets.Decorators;
using Platform.Exceptions;

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

namespace Platform.Data.Doublets
{
```

```
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 &amp;amp; TransactionIdCombined;
61          ///         }
62          ///     }
63          ///
64          ///     public UniqueTimestamp Timestamp
65          ///     {
66          ///         get
67          ///         {
68          ///             return (UniqueTimestamp)mask &amp;amp; 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.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly long Size = Structure<Transition>.Size;

        /// <summary>
        /// <para>
        /// The transaction id.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly ulong TransactionId;
        /// <summary>
        /// <para>
        /// The before.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly Link<ulong> Before;
        /// <summary>
        /// <para>
        /// The after.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly Link<ulong> After;
        /// <summary>
        /// <para>
        /// The timestamp.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly Timestamp Timestamp;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Transition"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="uniqueTimestampFactory">
        /// <para>A unique timestamp factory.</para>
        /// <para></para>
        /// </param>
        /// <param name="transactionId">
        /// <para>A transaction id.</para>
        /// <para></para>
        /// </param>
        /// <param name="before">
        /// <para>A before.</para>
        /// <para></para>
        /// </param>
        /// <param name="after">
        /// <para>A after.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↪  transactionId, Link<ulong> before, Link<ulong> after)
        {
            TransactionId = transactionId;
            Before = before;
            After = after;
            Timestamp = uniqueTimestampFactory.Create();
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Transition"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="uniqueTimestampFactory">
        /// <para>A unique timestamp factory.</para>
        /// <para></para>
        /// </param>
        /// <param name="transactionId">
        /// <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)
293         ///        будут записаны в лог.
294         ///
295         /// </remarks>
296         public class Transaction : DisposableBase
297         {
298             private readonly Queue<Transition> _transitions;
299             private readonly UInt64LinksTransactionsLayer _layer;
300             /// <summary>
301             /// <para>
302             /// Gets or sets the is committed value.
303             /// </para>
304             /// <para></para>
305             /// </summary>
306             public bool IsCommitted { get; private set; }
307             /// <summary>
308             /// <para>
309             /// Gets or sets the is reverted value.
310             /// </para>
311             /// <para></para>
312             /// </summary>
```

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

```
            /// <param name="transaction">
            /// <para>The transaction.</para>
            /// <para></para>
            /// </param>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            public static void SetCurrentTransaction(UInt64LinksTransactionsLayer layer,
            ↪  Transaction transaction)
            {
                layer._currentTransactionId = layer._lastCommitedTransactionId + 1;
                layer._currentTransactionTransitions = transaction._transitions;
                layer._currentTransaction = transaction;
            }

            /// <summary>
            /// <para>
            /// Ensures the transaction allows write operations using the specified transaction.
            /// </para>
            /// <para></para>
            /// </summary>
            /// <param name="transaction">
            /// <para>The transaction.</para>
            /// <para></para>
            /// </param>
            /// <exception cref="InvalidOperationException">
            /// <para>Transation is commited.</para>
            /// <para></para>
            /// </exception>
            /// <exception cref="InvalidOperationException">
            /// <para>Transation is reverted.</para>
            /// <para></para>
            /// </exception>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            public static void EnsureTransactionAllowsWriteOperations(Transaction transaction)
            {
                if (transaction.IsReverted)
                {
                    throw new InvalidOperationException("Transation is reverted.");
                }
                if (transaction.IsCommitted)
                {
                    throw new InvalidOperationException("Transation is commited.");
                }
            }

            /// <summary>
            /// <para>
            /// Disposes the manual.
            /// </para>
            /// <para></para>
            /// </summary>
            /// <param name="manual">
            /// <para>The manual.</para>
            /// <para></para>
            /// </param>
            /// <param name="wasDisposed">
            /// <para>The was disposed.</para>
            /// <para></para>
            /// </param>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            protected override void Dispose(bool manual, bool wasDisposed)
            {
                if (!wasDisposed && _layer != null && !_layer.Disposable.IsDisposed)
                {
                    if (!IsCommitted && !IsReverted)
                    {
                        Revert();
                    }
                    _layer.ResetCurrentTransation();
                }
            }
        }

        /// <summary>
        /// <para>
        /// The from seconds.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
468         public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);
469
470         private readonly string _logAddress;
471         private readonly FileStream _log;
472         private readonly Queue<Transition> _transitions;
473         private readonly UniqueTimestampFactory _uniqueTimestampFactory;
474         private Task _transitionsPusher;
475         private Transition _lastCommitedTransition;
476         private ulong _currentTransactionId;
477         private Queue<Transition> _currentTransactionTransitions;
478         private Transaction _currentTransaction;
479         private ulong _lastCommitedTransactionId;
480
481         /// <summary>
482         /// <para>
483         /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.
484         /// </para>
485         /// <para></para>
486         /// </summary>
487         /// <param name="links">
488         /// <para>A links.</para>
489         /// <para></para>
490         /// </param>
491         /// <param name="logAddress">
492         /// <para>A log address.</para>
493         /// <para></para>
494         /// </param>
495         /// <exception cref="ArgumentNullException">
496         /// <para></para>
497         /// <para></para>
498         /// </exception>
499         /// <exception cref="NotSupportedException">
500         /// <para>Database is damaged, autorecovery is not supported yet.</para>
501         /// <para></para>
502         /// </exception>
503         [MethodImpl(MethodImplOptions.AggressiveInlining)]
504         public UInt64LinksTransactionsLayer(ILinks<ulong> links, string logAddress)
505             : base(links)
506         {
507             if (string.IsNullOrWhiteSpace(logAddress))
508             {
509                 throw new ArgumentNullException(nameof(logAddress));
510             }
511             // В первой строке файла хранится последняя закоммиченную транзакцию.
512             // При запуске это используется для проверки удачного закрытия файла лога.
513             // In the first line of the file the last committed transaction is stored.
514             // On startup, this is used to check that the log file is successfully closed.
515             var lastCommitedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
516             var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
517             if (!lastCommitedTransition.Equals(lastWrittenTransition))
518             {
519                 Dispose();
520                 throw new NotSupportedException("Database is damaged, autorecovery is not
                    ↪  supported yet.");
521             }
522             if (lastCommitedTransition == default)
523             {
524                 FileHelpers.WriteFirst(logAddress, lastCommitedTransition);
525             }
526             _lastCommitedTransition = lastCommitedTransition;
527             // TODO: Think about a better way to calculate or store this value
528             var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
529             _lastCommitedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
                ↪  x.TransactionId) : 0;
530             _uniqueTimestampFactory = new UniqueTimestampFactory();
531             _logAddress = logAddress;
532             _log = FileHelpers.Append(logAddress);
533             _transitions = new Queue<Transition>();
534             _transitionsPusher = new Task(TransitionsPusher);
535             _transitionsPusher.Start();
536         }
537
538         /// <summary>
539         /// <para>
540         /// Gets the link value 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 list of ulong</para>
550             /// <para></para>
551             /// </returns>
552             [MethodImpl(MethodImplOptions.AggressiveInlining)]
553             public IList<ulong> GetLinkValue(ulong link) => _links.GetLink(link);
554
555             /// <summary>
556             /// <para>
557             /// Creates the restrictions.
558             /// </para>
559             /// <para></para>
560             /// </summary>
561             /// <param name="restrictions">
562             /// <para>The restrictions.</para>
563             /// <para></para>
564             /// </param>
565             /// <returns>
566             /// <para>The created link index.</para>
567             /// <para></para>
568             /// </returns>
569             [MethodImpl(MethodImplOptions.AggressiveInlining)]
570             public override ulong Create(IList<ulong> restrictions)
571             {
572                 var createdLinkIndex = _links.Create();
573                 var createdLink = new Link<ulong>(_links.GetLink(createdLinkIndex));
574                 CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
                    ↪  default, createdLink));
575                 return createdLinkIndex;
576             }
577
578             /// <summary>
579             /// <para>
580             /// Updates the restrictions.
581             /// </para>
582             /// <para></para>
583             /// </summary>
584             /// <param name="restrictions">
585             /// <para>The restrictions.</para>
586             /// <para></para>
587             /// </param>
588             /// <param name="substitution">
589             /// <para>The substitution.</para>
590             /// <para></para>
591             /// </param>
592             /// <returns>
593             /// <para>The link index.</para>
594             /// <para></para>
595             /// </returns>
596             [MethodImpl(MethodImplOptions.AggressiveInlining)]
597             public override ulong Update(IList<ulong> restrictions, IList<ulong> substitution)
598             {
599                 var linkIndex = restrictions[_constants.IndexPart];
600                 var beforeLink = new Link<ulong>(_links.GetLink(linkIndex));
601                 linkIndex = _links.Update(restrictions, substitution);
602                 var afterLink = new Link<ulong>(_links.GetLink(linkIndex));
603                 CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
                    ↪  beforeLink, afterLink));
604                 return linkIndex;
605             }
606
607             /// <summary>
608             /// <para>
609             /// Deletes the restrictions.
610             /// </para>
611             /// <para></para>
612             /// </summary>
613             /// <param name="restrictions">
614             /// <para>The restrictions.</para>
615             /// <para></para>
616             /// </param>
617             [MethodImpl(MethodImplOptions.AggressiveInlining)]
618             public override void Delete(IList<ulong> restrictions)
619             {
620                 var link = restrictions[_constants.IndexPart];
```

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

```
            _currentTransactionId = 0;
            _currentTransactionTransitions = null;
            _currentTransaction = null;
        }

        /// <summary>
        /// <para>
        /// Pushes the transitions.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private void PushTransitions()
        {
            if (_log == null || _transitions == null)
            {
                return;
            }
            for (var i = 0; i < _transitions.Count; i++)
            {
                var transition = _transitions.Dequeue();

                _log.Write(transition);
                _lastCommitedTransition = transition;
            }
        }

        /// <summary>
        /// <para>
        /// Transitionses the pusher.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private void TransitionsPusher()
        {
            while (!Disposable.IsDisposed && _transitionsPusher != null)
            {
                Thread.Sleep(DefaultPushDelay);
                PushTransitions();
            }
        }

        /// <summary>
        /// <para>
        /// Begins the transaction.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The transaction</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Transaction BeginTransaction() => new Transaction(this);

        /// <summary>
        /// <para>
        /// Disposes the transitions.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private void DisposeTransitions()
        {
            try
            {
                var pusher = _transitionsPusher;
                if (pusher != null)
                {
                    _transitionsPusher = null;
                    pusher.Wait();
                }
                if (_transitions != null)
                {
                    PushTransitions();
                }
                _log.DisposeIfPossible();
                FileHelpers.WriteFirst(_logAddress, _lastCommitedTransition);
```

```
            }
            catch (Exception ex)
            {
                ex.Ignore();
            }
        }

        #region DisposalBase

        /// <summary>
        /// <para>
        /// Disposes the manual.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="manual">
        /// <para>The manual.</para>
        /// <para></para>
        /// </param>
        /// <param name="wasDisposed">
        /// <para>The was disposed.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void Dispose(bool manual, bool wasDisposed)
        {
            if (!wasDisposed)
            {
                DisposeTransitions();
            }
            base.Dispose(manual, wasDisposed);
        }

        #endregion
    }
}
```

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

```
using System;
using Xunit;
using Platform.Reflection;
using Platform.Memory;
using Platform.Scopes;
using Platform.Data.Doublets.Memory.United.Generic;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the generic links tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public unsafe static class GenericLinksTests
    {
        /// <summary>
        /// <para>
        /// Tests that crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void CRUDTest()
        {
            Using<byte>(links => links.TestCRUDOperations());
            Using<ushort>(links => links.TestCRUDOperations());
            Using<uint>(links => links.TestCRUDOperations());
            Using<ulong>(links => links.TestCRUDOperations());
        }

        /// <summary>
        /// <para>
        /// Tests that raw numbers crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void RawNumbersCRUDTest()
        {
```

```
            Using<byte>(links => links.TestRawNumbersCRUDOperations());
            Using<ushort>(links => links.TestRawNumbersCRUDOperations());
            Using<uint>(links => links.TestRawNumbersCRUDOperations());
            Using<ulong>(links => links.TestRawNumbersCRUDOperations());
        }

        /// <summary>
        /// <para>
        /// Tests that multiple random creations and deletions test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void MultipleRandomCreationsAndDeletionsTest()
        {
            Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(16)); // Cannot use more because current implementation of tree cuts out 5 bits from the address space.
            Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
            Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
            Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
        }

        private static void Using<TLink>(Action<ILinks<TLink>> action)
        {
            using (var scope = new Scope<Types<HeapResizableDirectMemory, UnitedMemoryLinks<TLink>>>())
            {
                action(scope.Use<ILinks<TLink>>());
            }
        }
    }
}
```

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

```
using Xunit;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the links constants tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class LinksConstantsTests
    {
        /// <summary>
        /// <para>
        /// Tests that external references test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void ExternalReferencesTest()
        {
            LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue), (long.MaxValue + 1UL, ulong.MaxValue));

            //var minimum = new Hybrid<ulong>(0, isExternal: true);
            var minimum = new Hybrid<ulong>(1, isExternal: true);
            var maximum = new Hybrid<ulong>(long.MaxValue, isExternal: true);

            Assert.True(constants.IsExternalReference(minimum));
            Assert.True(constants.IsExternalReference(maximum));
        }
    }
}
```

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

```
using System.IO;
using Xunit;
using Platform.Singletons;
using Platform.Memory;
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          private static readonly LinksConstants<ulong> _constants =
            ↪  Default<LinksConstants<ulong>>.Instance;
18
19          /// <summary>
20          /// <para>
21          /// Tests that basic file mapped memory test.
22          /// </para>
23          /// <para></para>
24          /// </summary>
25          [Fact]
26          public static void BasicFileMappedMemoryTest()
27          {
28              var tempFilename = Path.GetTempFileName();
29              using (var memoryAdapter = new UInt64UnitedMemoryLinks(tempFilename))
30              {
31                  memoryAdapter.TestBasicMemoryOperations();
32              }
33              File.Delete(tempFilename);
34          }
35
36          /// <summary>
37          /// <para>
38          /// Tests that basic heap memory test.
39          /// </para>
40          /// <para></para>
41          /// </summary>
42          [Fact]
43          public static void BasicHeapMemoryTest()
44          {
45              using (var memory = new
                ↪  HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
46              using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
                ↪  UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
47              {
48                  memoryAdapter.TestBasicMemoryOperations();
49              }
50          }
51
52          private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
53          {
54              var link = memoryAdapter.Create();
55              memoryAdapter.Delete(link);
56          }
57
58          /// <summary>
59          /// <para>
60          /// Tests that nonexistent references heap memory test.
61          /// </para>
62          /// <para></para>
63          /// </summary>
64          [Fact]
65          public static void NonexistentReferencesHeapMemoryTest()
66          {
67              using (var memory = new
                ↪  HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
68              using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
                ↪  UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
69              {
70                  memoryAdapter.TestNonexistentReferences();
71              }
72          }
73
74          private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
75          {
76              var link = memoryAdapter.Create();
77              memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);
78              var resultLink = _constants.Null;
```

```
            memoryAdapter.Each(foundLink =>
            {
                resultLink = foundLink[_constants.IndexPart];
                return _constants.Break;
            }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
            Assert.True(resultLink == link);
            Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
            memoryAdapter.Delete(link);
        }
    }
}
```

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

```
using Xunit;
using Platform.Scopes;
using Platform.Memory;
using Platform.Data.Doublets.Decorators;
using Platform.Reflection;
using Platform.Data.Doublets.Memory.United.Generic;
using Platform.Data.Doublets.Memory.United.Specific;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the scope tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class ScopeTests
    {
        /// <summary>
        /// <para>
        /// Tests that single dependency test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void SingleDependencyTest()
        {
            using (var scope = new Scope())
            {
                scope.IncludeAssemblyOf<IMemory>();
                var instance = scope.Use<IDirectMemory>();
                Assert.IsType<HeapResizableDirectMemory>(instance);
            }
        }

        /// <summary>
        /// <para>
        /// Tests that cascade dependency test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void CascadeDependencyTest()
        {
            using (var scope = new Scope())
            {
                scope.Include<TemporaryFileMappedResizableDirectMemory>();
                scope.Include<UInt64UnitedMemoryLinks>();
                var instance = scope.Use<ILinks<ulong>>();
                Assert.IsType<UInt64UnitedMemoryLinks>(instance);
            }
        }

        /// <summary>
        /// <para>
        /// Tests that full auto resolution test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact(Skip = "Would be fixed later.")]
        public static void FullAutoResolutionTest()
        {
            using (var scope = new Scope(autoInclude: true, autoExplore: true))
            {
                var instance = scope.Use<UInt64Links>();
                Assert.IsType<UInt64Links>(instance);
```

```
            }
        }

        /// <summary>
        /// <para>
        /// Tests that type parameters test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void TypeParametersTest()
        {
            using (var scope = new Scope<Types<HeapResizableDirectMemory,
            ↪  UnitedMemoryLinks<ulong>>>())
            {
                var links = scope.Use<ILinks<ulong>>();
                Assert.IsType<UnitedMemoryLinks<ulong>>(links);
            }
        }
    }
}
```

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

```
using System;
using Xunit;
using Platform.Memory;
using Platform.Data.Doublets.Memory.Split.Generic;
using Platform.Data.Doublets.Memory;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the split memory generic links tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public unsafe static class SplitMemoryGenericLinksTests
    {
        /// <summary>
        /// <para>
        /// Tests that crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void CRUDTest()
        {
            Using<byte>(links => links.TestCRUDOperations());
            Using<ushort>(links => links.TestCRUDOperations());
            Using<uint>(links => links.TestCRUDOperations());
            Using<ulong>(links => links.TestCRUDOperations());
        }

        /// <summary>
        /// <para>
        /// Tests that raw numbers crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void RawNumbersCRUDTest()
        {
            UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
            UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
            UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
            UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
        }

        /// <summary>
        /// <para>
        /// Tests that multiple random creations and deletions test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void MultipleRandomCreationsAndDeletionsTest()
        {
```

```
            Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(16)); // Cannot use more because current implementation of tree cuts out 5 bits from the address space.
            Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
            Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
            Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultipleRandomCreationsAndDeletions(100));
        }

        private static void Using<TLink>(Action<ILinks<TLink>> action)
        {
            using (var dataMemory = new HeapResizableDirectMemory())
            using (var indexMemory = new HeapResizableDirectMemory())
            using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory))
            {
                action(memory);
            }
        }

        private static void UsingWithExternalReferences<TLink>(Action<ILinks<TLink>> action)
        {
            var contants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
            using (var dataMemory = new HeapResizableDirectMemory())
            using (var indexMemory = new HeapResizableDirectMemory())
            using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory, SplitMemoryLinks<TLink>.DefaultLinksSizeStep, contants))
            {
                action(memory);
            }
        }
    }
}
```

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

```
using System;
using Xunit;
using Platform.Memory;
using Platform.Data.Doublets.Memory.Split.Specific;
using TLink = System.UInt32;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the split memory int 32 links tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public unsafe static class SplitMemoryUInt32LinksTests
    {
        /// <summary>
        /// <para>
        /// Tests that crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void CRUDTest()
        {
            Using(links => links.TestCRUDOperations());
        }

        /// <summary>
        /// <para>
        /// Tests that raw numbers crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void RawNumbersCRUDTest()
        {
            UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
        }

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

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

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

```
1   using System;
2   using Xunit;
3   using Platform.Memory;
4   using Platform.Data.Doublets.Memory.Split.Specific;
5   using TLink = System.UInt64;
6
7   namespace Platform.Data.Doublets.Tests
8   {
9       /// <summary>
10      /// <para>
11      /// Represents the split memory int 64 links tests.
12      /// </para>
13      /// <para></para>
14      /// </summary>
15      public unsafe static class SplitMemoryUInt64LinksTests
16      {
17          /// <summary>
18          /// <para>
19          /// Tests that crud test.
20          /// </para>
21          /// <para></para>
22          /// </summary>
23          [Fact]
24          public static void CRUDTest()
25          {
26              Using(links => links.TestCRUDOperations());
27          }
28
29          /// <summary>
30          /// <para>
31          /// Tests that raw numbers crud test.
32          /// </para>
33          /// <para></para>
34          /// </summary>
35          [Fact]
36          public static void RawNumbersCRUDTest()
37          {
38              UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
39          }
40
41          /// <summary>
42          /// <para>
```

```
        /// Tests that multiple random creations and deletions test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void MultipleRandomCreationsAndDeletionsTest()
        {
            Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
            ↪  leRandomCreationsAndDeletions(500));
        }

        private static void Using(Action<ILinks<TLink>> action)
        {
            using (var dataMemory = new HeapResizableDirectMemory())
            using (var indexMemory = new HeapResizableDirectMemory())
            using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory))
            {
                action(memory);
            }
        }

        private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
        {
            var contants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
            using (var dataMemory = new HeapResizableDirectMemory())
            using (var indexMemory = new HeapResizableDirectMemory())
            using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
            ↪  UInt64SplitMemoryLinks.DefaultLinksSizeStep, contants))
            {
                action(memory);
            }
        }
    }
}
```

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

```
using System.Collections.Generic;
using Xunit;
using Platform.Ranges;
using Platform.Numbers;
using Platform.Random;
using Platform.Setters;
using Platform.Converters;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the test extensions.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class TestExtensions
    {
        /// <summary>
        /// <para>
        /// Tests the crud operations using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="T">
        /// <para>The .</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        public static void TestCRUDOperations<T>(this ILinks<T> links)
        {
            var constants = links.Constants;

            var equalityComparer = EqualityComparer<T>.Default;

            var zero = default(T);
            var one = Arithmetic.Increment(zero);

            // Create Link
            Assert.True(equalityComparer.Equals(links.Count(), zero));
```

```
            var setter = new Setter<T>(constants.Null);
            links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);

            Assert.True(equalityComparer.Equals(setter.Result, constants.Null));

            var linkAddress = links.Create();

            var link = new Link<T>(links.GetLink(linkAddress));

            Assert.True(link.Count == 3);
            Assert.True(equalityComparer.Equals(link.Index, linkAddress));
            Assert.True(equalityComparer.Equals(link.Source, constants.Null));
            Assert.True(equalityComparer.Equals(link.Target, constants.Null));

            Assert.True(equalityComparer.Equals(links.Count(), one));

            // Get first link
            setter = new Setter<T>(constants.Null);
            links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);

            Assert.True(equalityComparer.Equals(setter.Result, linkAddress));

            // Update link to reference itself
            links.Update(linkAddress, linkAddress, linkAddress);

            link = new Link<T>(links.GetLink(linkAddress));

            Assert.True(equalityComparer.Equals(link.Source, linkAddress));
            Assert.True(equalityComparer.Equals(link.Target, linkAddress));

            // Update link to reference null (prepare for delete)
            var updated = links.Update(linkAddress, constants.Null, constants.Null);

            Assert.True(equalityComparer.Equals(updated, linkAddress));

            link = new Link<T>(links.GetLink(linkAddress));

            Assert.True(equalityComparer.Equals(link.Source, constants.Null));
            Assert.True(equalityComparer.Equals(link.Target, constants.Null));

            // Delete link
            links.Delete(linkAddress);

            Assert.True(equalityComparer.Equals(links.Count(), zero));

            setter = new Setter<T>(constants.Null);
            links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);

            Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
        }

        /// <summary>
        /// <para>
        /// Tests the raw numbers crud operations using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="T">
        /// <para>The .</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        public static void TestRawNumbersCRUDOperations<T>(this ILinks<T> links)
        {
            // Constants
            var constants = links.Constants;
            var equalityComparer = EqualityComparer<T>.Default;

            var zero = default(T);
            var one = Arithmetic.Increment(zero);
            var two = Arithmetic.Increment(one);

            var h106E = new Hybrid<T>(106L, isExternal: true);
            var h107E = new Hybrid<T>(-char.ConvertFromUtf32(107)[0]);
            var h108E = new Hybrid<T>(-108L);
```

```
            Assert.Equal(106L, h106E.AbsoluteValue);
            Assert.Equal(107L, h107E.AbsoluteValue);
            Assert.Equal(108L, h108E.AbsoluteValue);

            // Create Link (External -> External)
            var linkAddress1 = links.Create();

            links.Update(linkAddress1, h106E, h108E);

            var link1 = new Link<T>(links.GetLink(linkAddress1));

            Assert.True(equalityComparer.Equals(link1.Source, h106E));
            Assert.True(equalityComparer.Equals(link1.Target, h108E));

            // Create Link (Internal -> External)
            var linkAddress2 = links.Create();

            links.Update(linkAddress2, linkAddress1, h108E);

            var link2 = new Link<T>(links.GetLink(linkAddress2));

            Assert.True(equalityComparer.Equals(link2.Source, linkAddress1));
            Assert.True(equalityComparer.Equals(link2.Target, h108E));

            // Create Link (Internal -> Internal)
            var linkAddress3 = links.Create();

            links.Update(linkAddress3, linkAddress1, linkAddress2);

            var link3 = new Link<T>(links.GetLink(linkAddress3));

            Assert.True(equalityComparer.Equals(link3.Source, linkAddress1));
            Assert.True(equalityComparer.Equals(link3.Target, linkAddress2));

            // Search for created link
            var setter1 = new Setter<T>(constants.Null);
            links.Each(h106E, h108E, setter1.SetAndReturnFalse);

            Assert.True(equalityComparer.Equals(setter1.Result, linkAddress1));

            // Search for nonexistent link
            var setter2 = new Setter<T>(constants.Null);
            links.Each(h106E, h107E, setter2.SetAndReturnFalse);

            Assert.True(equalityComparer.Equals(setter2.Result, constants.Null));

            // Update link to reference null (prepare for delete)
            var updated = links.Update(linkAddress3, constants.Null, constants.Null);

            Assert.True(equalityComparer.Equals(updated, linkAddress3));

            link3 = new Link<T>(links.GetLink(linkAddress3));

            Assert.True(equalityComparer.Equals(link3.Source, constants.Null));
            Assert.True(equalityComparer.Equals(link3.Target, constants.Null));

            // Delete link
            links.Delete(linkAddress3);

            Assert.True(equalityComparer.Equals(links.Count(), two));

            var setter3 = new Setter<T>(constants.Null);
            links.Each(constants.Any, constants.Any, setter3.SetAndReturnTrue);

            Assert.True(equalityComparer.Equals(setter3.Result, linkAddress2));
        }

        /// <summary>
        /// <para>
        /// Tests the multiple random creations and deletions using the specified links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
```

```
204         /// </param>
205         /// <param name="maximumOperationsPerCycle">
206         /// <para>The maximum operations per cycle.</para>
207         /// <para></para>
208         /// </param>
209         public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
            ↪ links, int maximumOperationsPerCycle)
210         {
211             var comparer = Comparer<TLink>.Default;
212             var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
213             var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
214             for (var N = 1; N < maximumOperationsPerCycle; N++)
215             {
216                 var random = new System.Random(N);
217                 var created = 0UL;
218                 var deleted = 0UL;
219                 for (var i = 0; i < N; i++)
220                 {
221                     var linksCount = addressToUInt64Converter.Convert(links.Count());
222                     var createPoint = random.NextBoolean();
223                     if (linksCount >= 2 && createPoint)
224                     {
225                         var linksAddressRange = new Range<ulong>(1, linksCount);
226                         TLink source = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
                            ↪ ddressRange));
227                         TLink target = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
                            ↪ ddressRange));
                            ↪ //-V3086
228                         var resultLink = links.GetOrCreate(source, target);
229                         if (comparer.Compare(resultLink,
                            ↪ uInt64ToAddressConverter.Convert(linksCount)) > 0)
230                         {
231                             created++;
232                         }
233                     }
234                     else
235                     {
236                         links.Create();
237                         created++;
238                     }
239                 }
240                 Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
241                 for (var i = 0; i < N; i++)
242                 {
243                     TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
244                     if (links.Exists(link))
245                     {
246                         links.Delete(link);
247                         deleted++;
248                     }
249                 }
250                 Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
251             }
252         }
253     }
254 }
```

1.124 ./csharp/Platform.Data.Doublets.Tests/Uint64LinksExtensionsTests.cs

```
1  using Platform.Data.Doublets.Memory;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Numbers.Raw;
4  using Platform.Memory;
5  using Platform.Numbers;
6  using Xunit;
7  using Xunit.Abstractions;
8  using TLink = System.UInt64;
9
10 namespace Platform.Data.Doublets.Tests
11 {
12     /// <summary>
13     /// <para>
14     /// Represents the uint 64 links extensions tests.
15     /// </para>
16     /// <para></para>
17     /// </summary>
18     public class Uint64LinksExtensionsTests
19     {
20         /// <summary>
21         /// <para>
```

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

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

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

```
    /// </para>
    /// <para></para>
    /// </summary>
    public unsafe static class UnitedMemoryUInt32LinksTests
    {
        /// <summary>
        /// <para>
        /// Tests that crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void CRUDTest()
        {
            Using(links => links.TestCRUDOperations());
        }

        /// <summary>
        /// <para>
        /// Tests that raw numbers crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void RawNumbersCRUDTest()
        {
            Using(links => links.TestRawNumbersCRUDOperations());
        }

        /// <summary>
        /// <para>
        /// Tests that multiple random creations and deletions test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void MultipleRandomCreationsAndDeletionsTest()
        {
            Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip」
            ↪ leRandomCreationsAndDeletions(100));
        }

        private static void Using(Action<ILinks<TLink>> action)
        {
            using (var scope = new Scope<Types<HeapResizableDirectMemory,
            ↪ UInt32UnitedMemoryLinks>>())
            {
                action(scope.Use<ILinks<TLink>>());
            }
        }
    }
}
```

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

```
using System;
using Xunit;
using Platform.Reflection;
using Platform.Memory;
using Platform.Scopes;
using Platform.Data.Doublets.Memory.United.Specific;
using TLink = System.UInt64;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the united memory int 64 links tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public unsafe static class UnitedMemoryUInt64LinksTests
    {
        /// <summary>
        /// <para>
        /// Tests that crud test.
        /// </para>
        /// <para></para>
        /// </summary>
        [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().TestMultip⌋
                ↪ leRandomCreationsAndDeletions(100));
53          }
54
55          private static void Using(Action<ILinks<TLink>> action)
56          {
57              using (var scope = new Scope<Types<HeapResizableDirectMemory,
                ↪ UInt64UnitedMemoryLinks>>())
58              {
59                  action(scope.Use<ILinks<TLink>>());
60              }
61          }
62      }
63  }
```

Index