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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// The target to match.
        /// </para>
        /// <para></para>
        /// </summary>
        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;
                }
            }
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksDecoratorBase"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected LinksDecoratorBase(ILinks<TLink> links) : base(links)
        {
            _constants = links.Constants;
            Facade = this;
        }

        /// <summary>
        /// <para>
        /// Counts 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 virtual TLink Count(IList<TLink> restrictions) => _links.Count(restrictions);

        /// <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 virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
        ↪  => _links.Each(handler, restrictions);

        /// <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 virtual TLink Create(IList<TLink> restrictions) => _links.Create(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 virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution) =>
        ↪  _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 virtual void Delete(IList<TLink> restrictions) => _links.Delete(restrictions);
    }
}
```

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

```
using System.Runtime.CompilerServices;
using Platform.Disposables;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
#pragma warning disable CA1063 // Implement IDisposable Correctly

namespace Platform.Data.Doublets.Decorators
{
    /// <summary>
    /// <para>
    /// Represents the links disposable decorator base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDecoratorBase{TLink}"/>
    /// <seealso cref="ILinks{TLink}"/>
    /// <seealso cref="System.IDisposable"/>
    public abstract class LinksDisposableDecoratorBase<TLink> : LinksDecoratorBase<TLink>,
    ↪  ILinks<TLink>, System.IDisposable
    {
        /// <summary>
        /// <para>
        /// Represents the disposable with multiple calls allowed.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <seealso cref="Disposable"/>
        protected class DisposableWithMultipleCallsAllowed : Disposable
        {
            /// <summary>
            /// <para>
```

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

            /// <summary>
            /// <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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        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>
```

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

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

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

1.10 ./csharp/Platform.Data.Doublets/Decorators/LinksUniquenessResolver.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 resolver.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDecoratorBase{TLink}"/>
    public class LinksUniquenessResolver<TLink> : LinksDecoratorBase<TLink>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

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

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

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

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

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

```
        /// <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
```

```
21      ///
22      /// Текущее теоретическое ограничение на индекс связи, из-за использования 5 бит в размере
        ↪  поддеревьев для AVL баланса и флагов нитей: 2 в степени(64 минус 5 равно 59 ) равно 576
        ↪  460 752 303 423 488
23      /// Желательно реализовать поддержку переключения между деревьями и битовыми индексами
        ↪  (битовыми строками) - вариант матрицы (выстраеваемой лениво).
24      ///
25      /// Решить отключать ли проверки при компиляции под Release. T.e. исключения будут
        ↪  выбрасываться только при #if DEBUG
26      /// </remarks>
27      public class UInt64Links : LinksDisposableDecoratorBase<ulong>
28      {
29          /// <summary>
30          /// <para>
31          /// Initializes a new <see cref="UInt64Links"/> instance.
32          /// </para>
33          /// <para></para>
34          /// </summary>
35          /// <param name="links">
36          /// <para>A links.</para>
37          /// <para></para>
38          /// </param>
39          [MethodImpl(MethodImplOptions.AggressiveInlining)]
40          public UInt64Links(ILinks<ulong> links) : base(links) { }
41
42          /// <summary>
43          /// <para>
44          /// Creates the restrictions.
45          /// </para>
46          /// <para></para>
47          /// </summary>
48          /// <param name="restrictions">
49          /// <para>The restrictions.</para>
50          /// <para></para>
51          /// </param>
52          /// <returns>
53          /// <para>The ulong</para>
54          /// <para></para>
55          /// </returns>
56          [MethodImpl(MethodImplOptions.AggressiveInlining)]
57          public override ulong Create(IList<ulong> restrictions) => _links.CreatePoint();
58
59          /// <summary>
60          /// <para>
61          /// Updates the restrictions.
62          /// </para>
63          /// <para></para>
64          /// </summary>
65          /// <param name="restrictions">
66          /// <para>The restrictions.</para>
67          /// <para></para>
68          /// </param>
69          /// <param name="substitution">
70          /// <para>The substitution.</para>
71          /// <para></para>
72          /// </param>
73          /// <returns>
74          /// <para>The ulong</para>
75          /// <para></para>
76          /// </returns>
77          [MethodImpl(MethodImplOptions.AggressiveInlining)]
78          public override ulong Update(IList<ulong> restrictions, IList<ulong> substitution)
79          {
80              var constants = _constants;
81              var indexPartConstant = constants.IndexPart;
82              var sourcePartConstant = constants.SourcePart;
83              var targetPartConstant = constants.TargetPart;
84              var nullConstant = constants.Null;
85              var itselfConstant = constants.Itself;
86              var existedLink = nullConstant;
87              var updatedLink = restrictions[indexPartConstant];
88              var newSource = substitution[sourcePartConstant];
89              var newTarget = substitution[targetPartConstant];
90              var links = _links;
91              if (newSource != itselfConstant && newTarget != itselfConstant)
92              {
93                  existedLink = links.SearchOrDefault(newSource, newTarget);
94              }
```

```
            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<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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        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) { }

        /// <summary>
        /// <para>
        /// The transition.
        /// </para>
        /// <para></para>
        /// </summary>
        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">
            /// <para>A before.</para>
            /// <para></para>
            /// </param>
            /// <param name="after">
            /// <para>A after.</para>
            /// <para></para>
            /// </param>
            public Transition(IList<TLink> before, IList<TLink> after)
            {
                Before = before;
                After = after;
            }
        }

        //public static readonly TLink NullConstant = Use<LinksConstants<TLink>>.Single.Null;
        //public static readonly IReadOnlyList<TLink> NullLink = new
        ↪  ReadOnlyCollection<TLink>(new List<TLink> { NullConstant, NullConstant, NullConstant
        ↪  });

        // TODO: Подумать о том, как реализовать древовидный Restriction и Substitution
        ↪  (Links-Expression)
        /// <summary>
        /// <para>
        /// Triggers the restriction.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="restriction">
        /// <para>The restriction.</para>
        /// <para></para>
        /// </param>
        /// <param name="matchedHandler">
        /// <para>The matched handler.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitution">
        /// <para>The substitution.</para>
        /// <para></para>
        /// </param>
        /// <param name="substitutedHandler">
        /// <para>The substituted handler.</para>
        /// <para></para>
```

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

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

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

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

```
                            after = _links.GetLink(substitution[0]);
                            _links.Update(linkToUpdate, constants.Null, constants.Null);
                            _links.Delete(linkToUpdate);
                        }
                    }
                    else if (substitution.Count == 3)
                    {
                        //Links.Update(after);
                    }
                    else
                    {
                        throw new NotSupportedException();
                    }
                    if (matchHandler != null)
                    {
                        return substitutionHandler(before, after);
                    }
                    return constants.Continue;
                }
                else
                {
                    throw new NotSupportedException();
                }
            }
        }

        /// <remarks>
        /// IList[IList[IList[T]]]
        /// |      |      |      |||
        /// |      |      |      ------ ||
        /// |      |      |       link  ||
        /// |      |      -------------- |
        /// |      |         change     |
        /// |       --------------------
        /// |         changes
        /// </remarks>
        public IList<IList<IList<TLink>>> Trigger(IList<TLink> condition, IList<TLink>
        ↪  substitution)
        {
            var changes = new List<IList<IList<TLink>>>();
            var @continue = _constants.Continue;
            Trigger(condition, AlwaysContinue, substitution, (before, after) =>
            {
                var change = new[] { before, after };
                changes.Add(change);
                return @continue;
            });
            return changes;
        }

        /// <summary>
        /// <para>
        /// Alwayses the continue using the specified link to match.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="linkToMatch">
        /// <para>The link to match.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        private TLink AlwaysContinue(IList<TLink> linkToMatch) => _constants.Continue;
    }
}
```

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

    /// <summury>
```

```
    /// <para>.</para>
    /// <para>.</para>
    /// </summury>
    /// <typeparam>
    /// <para>.</para>
    /// <para>.</para>
    /// </typeparam>
    public struct Doublet<T> : IEquatable<Doublet<T>>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<T> _equalityComparer =
        ↪  EqualityComparer<T>.Default;

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <typeparam name="T">
        /// <para>.</para>
        /// <para>.</para>
        /// </typeparam>
        public readonly T Source;

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <typeparam name="T">
        /// <para>.</para>
        /// <para>.</para>
        /// </typeparam>
        public readonly T Target;

        /// <summury>
        /// <para>.</para>
        /// <para>.</para>
        /// </summury>
        /// <typeparam name="T">
        /// <para>.</para>
        /// <para>.</para>
        /// </typeparam>
        /// <param name="source">
        /// <para>.</para>
        /// <para>.</para>
        /// </param>
        /// <param name="target">
        /// <para>.</para>
        /// <para>.</para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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>
```

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

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

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

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

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

```
    /// Defines the links.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ILinks{TLink, LinksConstants{TLink}}"/>
    public interface ILinks<TLink> : ILinks<TLink, LinksConstants<TLink>>
    {
    }
}
```

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

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.CompilerServices;
using Platform.Ranges;
using Platform.Collections.Arrays;
using Platform.Random;
using Platform.Setters;
using Platform.Converters;
using Platform.Numbers;
using Platform.Data.Exceptions;
using Platform.Data.Doublets.Decorators;

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

namespace Platform.Data.Doublets
{
    /// <summary>
    /// <para>
    /// Represents the links extensions.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class ILinksExtensions
    {
        /// <summary>
        /// <para>
        /// Runs the random creations 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="amountOfCreations">
        /// <para>The amount of creations.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void RunRandomCreations<TLink>(this ILinks<TLink> links, ulong
        ↪  amountOfCreations)
        {
            var random = RandomHelpers.Default;
            var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
            var uInt64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
            for (var i = 0UL; i < amountOfCreations; i++)
            {
                var linksAddressRange = new Range<ulong>(0,
                ↪  addressToUInt64Converter.Convert(links.Count()));
                var source =
                ↪  uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
                var target =
                ↪  uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
                links.GetOrCreate(source, target);
            }
        }

        /// <summary>
        /// <para>
        /// Runs the random searches 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="amountOfSearches">
        /// <para>The amount of searches.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void RunRandomSearches<TLink>(this ILinks<TLink> links, ulong
        ↪  amountOfSearches)
        {
            var random = RandomHelpers.Default;
            var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
            var uInt64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
            for (var i = 0UL; i < amountOfSearches; i++)
            {
                var linksAddressRange = new Range<ulong>(0,
                ↪  addressToUInt64Converter.Convert(links.Count()));
                var source =
                ↪  uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
                var target =
                ↪  uInt64ToAddressConverter.Convert(random.NextUInt64(linksAddressRange));
                links.SearchOrDefault(source, target);
            }
        }

        /// <summary>
        /// <para>
        /// Runs the random 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>
        /// </param>
        /// <param name="amountOfDeletions">
        /// <para>The amount of deletions.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void RunRandomDeletions<TLink>(this ILinks<TLink> links, ulong
        ↪  amountOfDeletions)
        {
            var random = RandomHelpers.Default;
            var addressToUInt64Converter = UncheckedConverter<TLink, ulong>.Default;
            var uInt64ToAddressConverter = UncheckedConverter<ulong, TLink>.Default;
            var linksCount = addressToUInt64Converter.Convert(links.Count());
            var min = amountOfDeletions > linksCount ? 0UL : linksCount - amountOfDeletions;
            for (var i = 0UL; 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>
```

```
137         /// </summary>
138         /// <typeparam name="TLink">
139         /// <para>The link.</para>
140         /// <para></para>
141         /// </typeparam>
142         /// <param name="links">
143         /// <para>The links.</para>
144         /// <para></para>
145         /// </param>
146         /// <param name="linkToDelete">
147         /// <para>The link to delete.</para>
148         /// <para></para>
149         /// </param>
150         [MethodImpl(MethodImplOptions.AggressiveInlining)]
151         public static void Delete<TLink>(this ILinks<TLink> links, TLink linkToDelete)
152         {
153             if (links.Exists(linkToDelete))
154             {
155                 links.EnforceResetValues(linkToDelete);
156                 links.Delete(new LinkAddress<TLink>(linkToDelete));
157             }
158         }
159 
160         /// <remarks>
161         /// TODO: Возможно есть очень простой способ это сделать.
162         /// (Например просто удалить файл, или изменить его размер таким образом,
163         /// чтобы удалился весь контент)
164         /// Например через _header->AllocatedLinks в ResizableDirectMemoryLinks
165         /// </remarks>
166         [MethodImpl(MethodImplOptions.AggressiveInlining)]
167         public static void DeleteAll<TLink>(this ILinks<TLink> links)
168         {
169             var equalityComparer = EqualityComparer<TLink>.Default;
170             var comparer = Comparer<TLink>.Default;
171             for (var i = links.Count(); comparer.Compare(i, default) > 0; i =
                ↪  Arithmetic.Decrement(i))
172             {
173                 links.Delete(i);
174                 if (!equalityComparer.Equals(links.Count(), Arithmetic.Decrement(i)))
175                 {
176                     i = links.Count();
177                 }
178             }
179         }
180 
181         /// <summary>
182         /// <para>
183         /// Firsts the links.
184         /// </para>
185         /// <para></para>
186         /// </summary>
187         /// <typeparam name="TLink">
188         /// <para>The link.</para>
189         /// <para></para>
190         /// </typeparam>
191         /// <param name="links">
192         /// <para>The links.</para>
193         /// <para></para>
194         /// </param>
195         /// <exception cref="InvalidOperationException">
196         /// <para>В процессе поиска по хранилищу не было найдено связей.</para>
197         /// <para></para>
198         /// </exception>
199         /// <exception cref="InvalidOperationException">
200         /// <para>В хранилище нет связей.</para>
201         /// <para></para>
202         /// </exception>
203         /// <returns>
204         /// <para>The first link.</para>
205         /// <para></para>
206         /// </returns>
207         [MethodImpl(MethodImplOptions.AggressiveInlining)]
208         public static TLink First<TLink>(this ILinks<TLink> links)
209         {
210             TLink firstLink = default;
211             var equalityComparer = EqualityComparer<TLink>.Default;
212             if (equalityComparer.Equals(links.Count(), default))
213             {
```

```
214                 throw new InvalidOperationException("В хранилище нет связей.");
215             }
216             links.Each(links.Constants.Any, links.Constants.Any, link =>
217             {
218                 firstLink = link[links.Constants.IndexPart];
219                 return links.Constants.Break;
220             });
221             if (equalityComparer.Equals(firstLink, default))
222             {
223                 throw new InvalidOperationException("В процессе поиска по хранилищу не было
                    ↪  найдено связей.");
224             }
225             return firstLink;
226         }
227 
228         /// <summary>
229         /// <para>
230         /// Singles the or default using the specified links.
231         /// </para>
232         /// <para></para>
233         /// </summary>
234         /// <typeparam name="TLink">
235         /// <para>The link.</para>
236         /// <para></para>
237         /// </typeparam>
238         /// <param name="links">
239         /// <para>The links.</para>
240         /// <para></para>
241         /// </param>
242         /// <param name="query">
243         /// <para>The query.</para>
244         /// <para></para>
245         /// </param>
246         /// <returns>
247         /// <para>The result.</para>
248         /// <para></para>
249         /// </returns>
250         [MethodImpl(MethodImplOptions.AggressiveInlining)]
251         public static IList<TLink> SingleOrDefault<TLink>(this ILinks<TLink> links, IList<TLink>
            ↪  query)
252         {
253             IList<TLink> result = null;
254             var count = 0;
255             var constants = links.Constants;
256             var @continue = constants.Continue;
257             var @break = constants.Break;
258             links.Each(linkHandler, query);
259             return result;
260 
261             TLink linkHandler(IList<TLink> link)
262             {
263                 if (count == 0)
264                 {
265                     result = link;
266                     count++;
267                     return @continue;
268                 }
269                 else
270                 {
271                     result = null;
272                     return @break;
273                 }
274             }
275         }
276 
277         #region Paths
278 
279         /// <remarks>
280         /// TODO: Как так? Как то что ниже может быть корректно?
281         /// Скорее всего практически не применимо
282         /// Предполагалось, что можно было конвертировать формируемый в проходе через
            ↪  SequenceWalker
283         /// Stack в конкретный путь из Source, Target до связи, но это не всегда так.
284         /// TODO: Возможно нужен метод, который именно выбрасывает исключения (EnsurePathExists)
285         /// </remarks>
286         [MethodImpl(MethodImplOptions.AggressiveInlining)]
287         public static bool CheckPathExistance<TLink>(this ILinks<TLink> links, params TLink[]
            ↪  path)
288         {
```

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

```
        /// <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];

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

        /// <summary>
        /// Возвращает индекс конечной (Target) связи для указанной связи.
        /// </summary>
        /// <param name="links">Хранилище связей.</param>
```

```
430         /// <param name="link">Связь представленная списком, состоящим из её адреса и
            ↪ содержимого.</param>
431         /// <returns>Индекс конечной связи для указанной связи.</returns>
432         [MethodImpl(MethodImplOptions.AggressiveInlining)]
433         public static TLink GetTarget<TLink>(this ILinks<TLink> links, IList<TLink> link) =>
            ↪ link[links.Constants.TargetPart];
434
435         /// <summary>
436         /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
            ↪ (handler) для каждой подходящей связи.
437         /// </summary>
438         /// <param name="links">Хранилище связей.</param>
439         /// <param name="handler">Обработчик каждой подходящей связи.</param>
440         /// <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>
441         /// <returns>True, в случае если проход по связям не был прерван и False в обратном
            ↪ случае.</returns>
442         [MethodImpl(MethodImplOptions.AggressiveInlining)]
443         public static bool Each<TLink>(this ILinks<TLink> links, Func<IList<TLink>, TLink>
            ↪ handler, params TLink[] restrictions)
444             => EqualityComparer<TLink>.Default.Equals(links.Each(handler, restrictions),
                ↪ links.Constants.Continue);
445
446         /// <summary>
447         /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
            ↪ (handler) для каждой подходящей связи.
448         /// </summary>
449         /// <param name="links">Хранилище связей.</param>
450         /// <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>
451         /// <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>
452         /// <param name="handler">Обработчик каждой подходящей связи.</param>
453         /// <returns>True, в случае если проход по связям не был прерван и False в обратном
            ↪ случае.</returns>
454         [MethodImpl(MethodImplOptions.AggressiveInlining)]
455         public static bool Each<TLink>(this ILinks<TLink> links, TLink source, TLink target,
            ↪ Func<TLink, bool> handler)
456         {
457             var constants = links.Constants;
458             return links.Each(link => handler(link[constants.IndexPart]) ? constants.Continue :
                ↪ constants.Break, constants.Any, source, target);
459         }
460
461         /// <summary>
462         /// Выполняет проход по всем связям, соответствующим шаблону, вызывая обработчик
            ↪ (handler) для каждой подходящей связи.
463         /// </summary>
464         /// <param name="links">Хранилище связей.</param>
465         /// <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>
466         /// <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>
467         /// <param name="handler">Обработчик каждой подходящей связи.</param>
468         /// <returns>True, в случае если проход по связям не был прерван и False в обратном
            ↪ случае.</returns>
469         [MethodImpl(MethodImplOptions.AggressiveInlining)]
470         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);
471
472         /// <summary>
473         /// <para>
474         /// Alls the links.
475         /// </para>
476         /// <para></para>
477         /// </summary>
478         /// <typeparam name="TLink">
479         /// <para>The link.</para>
480         /// <para></para>
481         /// </typeparam>
```

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

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

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

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

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

        /// <param name="links">Хранилище связей.</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnsureDoesNotExists<TLink>(this ILinks<TLink> links, TLink source,
        ↪  TLink target)
        {
```

```
            if (links.Exists(source, target))
            {
                throw new LinkWithSameValueAlreadyExistsException();
            }
        }

        /// <param name="links">Хранилище связей.</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnsureNoUsages<TLink>(this ILinks<TLink> links, TLink link)
        {
            if (links.HasUsages(link))
            {
                throw new ArgumentLinkHasDependenciesException<TLink>(link);
            }
        }

        /// <param name="links">Хранилище связей.</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnsureCreated<TLink>(this ILinks<TLink> links, params TLink[]
        ↪  addresses) => links.EnsureCreated(links.Create, addresses);

        /// <param name="links">Хранилище связей.</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnsurePointsCreated<TLink>(this ILinks<TLink> links, params TLink[]
        ↪  addresses) => links.EnsureCreated(links.CreatePoint, addresses);

        /// <param name="links">Хранилище связей.</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void EnsureCreated<TLink>(this ILinks<TLink> links, Func<TLink> creator,
        ↪  params TLink[] addresses)
        {
            var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
            var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
            var nonExistentAddresses = new HashSet<TLink>(addresses.Where(x =>
            ↪  !links.Exists(x)));
            if (nonExistentAddresses.Count > 0)
            {
                var max = nonExistentAddresses.Max();
                max = uInt64ToAddressConverter.Convert(System.Math.Min(addressToUInt64Converter.
                ↪  Convert(max),
                ↪  addressToUInt64Converter.Convert(links.Constants.InternalReferencesRange.Max
                ↪  imum)));
                var createdLinks = new List<TLink>();
                var equalityComparer = EqualityComparer<TLink>.Default;
                TLink createdLink = creator();
                while (!equalityComparer.Equals(createdLink, max))
                {
                    createdLinks.Add(createdLink);
                }
                for (var i = 0; i < createdLinks.Count; i++)
                {
                    if (!nonExistentAddresses.Contains(createdLinks[i]))
                    {
                        links.Delete(createdLinks[i]);
                    }
                }
            }
        }

        #endregion

        /// <param name="links">Хранилище связей.</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TLink CountUsages<TLink>(this ILinks<TLink> links, TLink link)
        {
            var constants = links.Constants;
            var values = links.GetLink(link);
            TLink usagesAsSource = links.Count(new Link<TLink>(constants.Any, link,
            ↪  constants.Any));
            var equalityComparer = EqualityComparer<TLink>.Default;
            if (equalityComparer.Equals(values[constants.SourcePart], link))
            {
                usagesAsSource = Arithmetic<TLink>.Decrement(usagesAsSource);
            }
            TLink usagesAsTarget = links.Count(new Link<TLink>(constants.Any, constants.Any,
            ↪  link));
            if (equalityComparer.Equals(values[constants.TargetPart], link))
            {
```

```
844                 usagesAsTarget = Arithmetic<TLink>.Decrement(usagesAsTarget);
845             }
846             return Arithmetic<TLink>.Add(usagesAsSource, usagesAsTarget);
847         }
848 
849         /// <param name="links">Хранилище связей.</param>
850         [MethodImpl(MethodImplOptions.AggressiveInlining)]
851         public static bool HasUsages<TLink>(this ILinks<TLink> links, TLink link) =>
            ↪ Comparer<TLink>.Default.Compare(links.CountUsages(link), default) > 0;
852 
853         /// <param name="links">Хранилище связей.</param>
854         [MethodImpl(MethodImplOptions.AggressiveInlining)]
855         public static bool Equals<TLink>(this ILinks<TLink> links, TLink link, TLink source,
            ↪ TLink target)
856         {
857             var constants = links.Constants;
858             var values = links.GetLink(link);
859             var equalityComparer = EqualityComparer<TLink>.Default;
860             return equalityComparer.Equals(values[constants.SourcePart], source) &&
                ↪ equalityComparer.Equals(values[constants.TargetPart], target);
861         }
862 
863         /// <summary>
864         /// Выполняет поиск связи с указанными Source (началом) и Target (концом).
865         /// </summary>
866         /// <param name="links">Хранилище связей.</param>
867         /// <param name="source">Индекс связи, которая является началом для искомой
            ↪ связи.</param>
868         /// <param name="target">Индекс связи, которая является концом для искомой связи.</param>
869         /// <returns>Индекс искомой связи с указанными Source (началом) и Target
            ↪ (концом).</returns>
870         [MethodImpl(MethodImplOptions.AggressiveInlining)]
871         public static TLink SearchOrDefault<TLink>(this ILinks<TLink> links, TLink source, TLink
            ↪ target)
872         {
873             var contants = links.Constants;
874             var setter = new Setter<TLink, TLink>(contants.Continue, contants.Break, default);
875             links.Each(setter.SetFirstAndReturnFalse, contants.Any, source, target);
876             return setter.Result;
877         }
878 
879         /// <param name="links">Хранилище связей.</param>
880         [MethodImpl(MethodImplOptions.AggressiveInlining)]
881         public static TLink Create<TLink>(this ILinks<TLink> links) => links.Create(null);
882 
883         /// <param name="links">Хранилище связей.</param>
884         [MethodImpl(MethodImplOptions.AggressiveInlining)]
885         public static TLink CreatePoint<TLink>(this ILinks<TLink> links)
886         {
887             var link = links.Create();
888             return links.Update(link, link, link);
889         }
890 
891         /// <param name="links">Хранилище связей.</param>
892         [MethodImpl(MethodImplOptions.AggressiveInlining)]
893         public static TLink CreateAndUpdate<TLink>(this ILinks<TLink> links, TLink source, TLink
            ↪ target) => links.Update(links.Create(), source, target);
894 
895         /// <summary>
896         /// Обновляет связь с указанными началом (Source) и концом (Target)
897         /// на связь с указанными началом (NewSource) и концом (NewTarget).
898         /// </summary>
899         /// <param name="links">Хранилище связей.</param>
900         /// <param name="link">Индекс обновляемой связи.</param>
901         /// <param name="newSource">Индекс связи, которая является началом связи, на которую
            ↪ выполняется обновление.</param>
902         /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
            ↪ выполняется обновление.</param>
903         /// <returns>Индекс обновлённой связи.</returns>
904         [MethodImpl(MethodImplOptions.AggressiveInlining)]
905         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));
906 
907         /// <summary>
908         /// Обновляет связь с указанными началом (Source) и концом (Target)
909         /// на связь с указанными началом (NewSource) и концом (NewTarget).
910         /// </summary>
```

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

```
983             /// <param name="source">Индекс связи, которая является началом на создаваемой
                ↪  связи.</param>
984             /// <param name="target">Индекс связи, которая является концом для создаваемой
                ↪  связи.</param>
985             /// <returns>Индекс связи, с указанным Source (началом) и Target (концом)</returns>
986             [MethodImpl(MethodImplOptions.AggressiveInlining)]
987             public static TLink GetOrCreate<TLink>(this ILinks<TLink> links, TLink source, TLink
                ↪  target)
988             {
989                 var link = links.SearchOrDefault(source, target);
990                 if (EqualityComparer<TLink>.Default.Equals(link, default))
991                 {
992                     link = links.CreateAndUpdate(source, target);
993                 }
994                 return link;
995             }
996
997             /// <summary>
998             /// Обновляет связь с указанными началом (Source) и концом (Target)
999             /// на связь с указанными началом (NewSource) и концом (NewTarget).
1000            /// </summary>
1001            /// <param name="links">Хранилище связей.</param>
1002            /// <param name="source">Индекс связи, которая является началом обновляемой
                ↪  связи.</param>
1003            /// <param name="target">Индекс связи, которая является концом обновляемой связи.</param>
1004            /// <param name="newSource">Индекс связи, которая является началом связи, на которую
                ↪  выполняется обновление.</param>
1005            /// <param name="newTarget">Индекс связи, которая является концом связи, на которую
                ↪  выполняется обновление.</param>
1006            /// <returns>Индекс обновлённой связи.</returns>
1007            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1008            public static TLink UpdateOrCreateOrGet<TLink>(this ILinks<TLink> links, TLink source,
                ↪  TLink target, TLink newSource, TLink newTarget)
1009            {
1010                var equalityComparer = EqualityComparer<TLink>.Default;
1011                var link = links.SearchOrDefault(source, target);
1012                if (equalityComparer.Equals(link, default))
1013                {
1014                    return links.CreateAndUpdate(newSource, newTarget);
1015                }
1016                if (equalityComparer.Equals(newSource, source) && equalityComparer.Equals(newTarget,
                    ↪  target))
1017                {
1018                    return link;
1019                }
1020                return links.Update(link, newSource, newTarget);
1021            }
1022
1023            /// <summary>Удаляет связь с указанными началом (Source) и концом (Target).</summary>
1024            /// <param name="links">Хранилище связей.</param>
1025            /// <param name="source">Индекс связи, которая является началом удаляемой связи.</param>
1026            /// <param name="target">Индекс связи, которая является концом удаляемой связи.</param>
1027            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1028            public static TLink DeleteIfExists<TLink>(this ILinks<TLink> links, TLink source, TLink
                ↪  target)
1029            {
1030                var link = links.SearchOrDefault(source, target);
1031                if (!EqualityComparer<TLink>.Default.Equals(link, default))
1032                {
1033                    links.Delete(link);
1034                    return link;
1035                }
1036                return default;
1037            }
1038
1039            /// <summary>Удаляет несколько связей.</summary>
1040            /// <param name="links">Хранилище связей.</param>
1041            /// <param name="deletedLinks">Список адресов связей к удалению.</param>
1042            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1043            public static void DeleteMany<TLink>(this ILinks<TLink> links, IList<TLink> deletedLinks)
1044            {
1045                for (int i = 0; i < deletedLinks.Count; i++)
1046                {
1047                    links.Delete(deletedLinks[i]);
1048                }
1049            }
1050
```

```
        /// <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 any = links.Constants.Any;
            var usagesAsSourceQuery = new Link<TLink>(any, linkIndex, any);
            links.DeleteByQuery(usagesAsSourceQuery);
            var usagesAsTargetQuery = new Link<TLink>(any, any, linkIndex);
            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))
            {
                links.ResetValues(linkIndex);
            }
        }

        /// <summary>
        /// Merging two usages graphs, all children of old link moved to be children of new link or deleted.
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static TLink MergeUsages<TLink>(this ILinks<TLink> links, TLink oldLinkIndex, TLink newLinkIndex)
        {
            var addressToInt64Converter = CheckedConverter<TLink, long>.Default;
            var equalityComparer = EqualityComparer<TLink>.Default;
            if (!equalityComparer.Equals(oldLinkIndex, newLinkIndex))
            {
```

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

```
}
```

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

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

namespace Platform.Data.Doublets
{
    /// <summary>
    /// <para>
    /// Defines the synchronized links.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ISynchronizedLinks{TLink, ILinks{TLink}, LinksConstants{TLink}}"/>
    /// <seealso cref="ILinks{TLink}"/>
    public interface ISynchronizedLinks<TLink> : ISynchronizedLinks<TLink, ILinks<TLink>,
    ↪  LinksConstants<TLink>>, ILinks<TLink>
    {
    }
}
```

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

```
using Platform.Collections.Lists;
using Platform.Exceptions;
using Platform.Ranges;
using Platform.Singletons;
using System;
using System.Collections;
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
{
    /// <summary>
    /// Структура описывающая уникальную связь.
    /// </summary>
    public struct Link<TLink> : IEquatable<Link<TLink>>, IReadOnlyList<TLink>, IList<TLink>
    {
        /// <summary>
        /// <para>
        /// The link.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly Link<TLink> Null = new Link<TLink>();

        /// <summary>
        /// <para>
        /// The instance.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly LinksConstants<TLink> _constants =
        ↪  Default<LinksConstants<TLink>>.Instance;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// The length.
        /// </para>
        /// <para></para>
        /// </summary>
        private const int Length = 3;

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

```
        public readonly TLink Index;
        /// <summary>
        /// <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>
```

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

```
        /// <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>
```

```
287             /// <para>The bool</para>
288             /// <para></para>
289             /// </returns>
290             [MethodImpl(MethodImplOptions.AggressiveInlining)]
291             public override bool Equals(object other) => other is Link<TLink> &&
                ↪  Equals((Link<TLink>)other);
292
293             /// <summary>
294             /// <para>
295             /// Determines whether this instance equals.
296             /// </para>
297             /// <para></para>
298             /// </summary>
299             /// <param name="other">
300             /// <para>The other.</para>
301             /// <para></para>
302             /// </param>
303             /// <returns>
304             /// <para>The bool</para>
305             /// <para></para>
306             /// </returns>
307             [MethodImpl(MethodImplOptions.AggressiveInlining)]
308             public bool Equals(Link<TLink> other) => _equalityComparer.Equals(Index, other.Index)
309                                                   && _equalityComparer.Equals(Source, other.Source)
310                                                   && _equalityComparer.Equals(Target, other.Target);
311
312             /// <summary>
313             /// <para>
314             /// Returns the string using the specified index.
315             /// </para>
316             /// <para></para>
317             /// </summary>
318             /// <param name="index">
319             /// <para>The index.</para>
320             /// <para></para>
321             /// </param>
322             /// <param name="source">
323             /// <para>The source.</para>
324             /// <para></para>
325             /// </param>
326             /// <param name="target">
327             /// <para>The target.</para>
328             /// <para></para>
329             /// </param>
330             /// <returns>
331             /// <para>The string</para>
332             /// <para></para>
333             /// </returns>
334             [MethodImpl(MethodImplOptions.AggressiveInlining)]
335             public static string ToString(TLink index, TLink source, TLink target) => $"({index}:
                ↪  {source}->{target})";
336
337             /// <summary>
338             /// <para>
339             /// Returns the string using the specified source.
340             /// </para>
341             /// <para></para>
342             /// </summary>
343             /// <param name="source">
344             /// <para>The source.</para>
345             /// <para></para>
346             /// </param>
347             /// <param name="target">
348             /// <para>The target.</para>
349             /// <para></para>
350             /// </param>
351             /// <returns>
352             /// <para>The string</para>
353             /// <para></para>
354             /// </returns>
355             [MethodImpl(MethodImplOptions.AggressiveInlining)]
356             public static string ToString(TLink source, TLink target) => $"({source}->{target})";
357
358             [MethodImpl(MethodImplOptions.AggressiveInlining)]
359             public static implicit operator TLink[](Link<TLink> link) => link.ToArray();
360
361             [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
362         public static implicit operator Link<TLink>(TLink[] linkArray) => new
            ↪  Link<TLink>(linkArray);
363
364         /// <summary>
365         /// <para>
366         /// Returns the string.
367         /// </para>
368         /// <para></para>
369         /// </summary>
370         /// <returns>
371         /// <para>The string</para>
372         /// <para></para>
373         /// </returns>
374         [MethodImpl(MethodImplOptions.AggressiveInlining)]
375         public override string ToString() => _equalityComparer.Equals(Index, _constants.Null) ?
            ↪  ToString(Source, Target) : ToString(Index, Source, Target);
376
377         #region IList
378
379         /// <summary>
380         /// <para>
381         /// Gets the count value.
382         /// </para>
383         /// <para></para>
384         /// </summary>
385         public int Count
386         {
387             [MethodImpl(MethodImplOptions.AggressiveInlining)]
388             get => Length;
389         }
390
391         /// <summary>
392         /// <para>
393         /// Gets the is read only value.
394         /// </para>
395         /// <para></para>
396         /// </summary>
397         public bool IsReadOnly
398         {
399             [MethodImpl(MethodImplOptions.AggressiveInlining)]
400             get => true;
401         }
402
403         /// <summary>
404         /// <para>
405         /// The not supported exception.
406         /// </para>
407         /// <para></para>
408         /// </summary>
409         public TLink this[int index]
410         {
411             [MethodImpl(MethodImplOptions.AggressiveInlining)]
412             get
413             {
414                 Ensure.OnDebug.ArgumentInRange(index, new Range<int>(0, Length - 1),
                    ↪  nameof(index));
415                 if (index == _constants.IndexPart)
416                 {
417                     return Index;
418                 }
419                 if (index == _constants.SourcePart)
420                 {
421                     return Source;
422                 }
423                 if (index == _constants.TargetPart)
424                 {
425                     return Target;
426                 }
427                 throw new NotSupportedException(); // Impossible path due to
                    ↪  Ensure.ArgumentInRange
428             }
429             [MethodImpl(MethodImplOptions.AggressiveInlining)]
430             set => throw new NotSupportedException();
431         }
432
433         /// <summary>
434         /// <para>
435         /// Gets the enumerator.
436         /// </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>
```

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

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

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

```
        /// <para></para>
        /// </typeparam>
        /// <param name="link">
        /// <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>
```

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

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

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets.Memory
4  {
5      /// <summary>
6      /// <para>
7      /// The index tree type enum.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public enum IndexTreeType
12     {
13         /// <summary>
```

```
        /// <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>>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        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="ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> 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 ExternalLinksRecursionlessSizeBalancedTreeMethodsBase(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>
        /// Determines whether this instance first is to the right 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 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>
```

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

```
        /// <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);
        }

        /// <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)
        {
            ref var firstLink = ref GetLinkDataPartReference(first);
            ref var secondLink = ref GetLinkDataPartReference(second);
            return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪  secondLink.Source, secondLink.Target);
        }

        /// <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>
```

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

```
                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);
        }
    }
}
```

1.30 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/ExternalLinksSizeBalancedTreeMethodsBase.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 external links size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="SizeBalancedTreeMethods{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class ExternalLinksSizeBalancedTreeMethodsBase<TLink> :
    ↪  SizeBalancedTreeMethods<TLink>, ILinksTreeMethods<TLink>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        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="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)]
```

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

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

```
        /// <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);
        }

        /// <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)
        {
            ref var firstLink = ref GetLinkDataPartReference(first);
            ref var secondLink = ref GetLinkDataPartReference(second);
            return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
             ↪  secondLink.Source, secondLink.Target);
        }

        /// <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 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) { }

        /// <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>
```

```
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>
```

```
        /// <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).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() => GetHeaderReference().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 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>
```

```
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">
```

```
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public ExternalLinksSourcesSizeBalancedTreeMethods(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)]
        protected override TLink GetRight(TLink node) =>
        ↪  GetLinkIndexPartReference(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) =>
        ↪  GetLinkIndexPartReference(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) =>
        ↪  GetLinkIndexPartReference(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) =>
        ↪  GetLinkIndexPartReference(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) =>
        ↪  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">
```

```
        /// <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)]
```

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

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

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

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

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

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

```
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.
180             /// </para>
181             /// <para></para>
182             /// </summary>
183             /// <returns>
184             /// <para>The link</para>
185             /// <para></para>
186             /// </returns>
187             [MethodImpl(MethodImplOptions.AggressiveInlining)]
188             protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
189
190             /// <summary>
191             /// <para>
192             /// Gets the base part value using the specified link.
193             /// </para>
```

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

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

        /// <summary>
        /// <para>
        /// Determines whether this instance first is to the right of second.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="firstSource">
        /// <para>The first source.</para>
        /// <para></para>
        /// </param>
        /// <param name="firstTarget">
        /// <para>The first target.</para>
        /// <para></para>
        /// </param>
        /// <param name="secondSource">
        /// <para>The second source.</para>
        /// <para></para>
        /// </param>
        /// <param name="secondTarget">
        /// <para>The second target.</para>
        /// <para></para>
        /// </param>
        /// <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 GetLinkIndexPartReference(node);
            link.LeftAsTarget = Zero;
            link.RightAsTarget = Zero;
            link.SizeAsTarget = Zero;
        }
    }
}
```

1.35 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/InternalLinksRecursionlessSizeBalancedTreeMethod

```
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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        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">
        /// <para>A links index parts.</para>
        /// <para></para>
        /// </param>
        /// <param name="header">
        /// <para>A header.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected InternalLinksRecursionlessSizeBalancedTreeMethodsBase(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>
        /// <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>
        /// </summary>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A list of t link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual IList<TLink> GetLinkValues(TLink linkIndex)
        {
            ref var link = ref GetLinkDataPartReference(linkIndex);
            return new Link<TLink>(linkIndex, link.Source, link.Target);
        }

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

```
282
283         /// <summary>
284         /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
            ↪  (концом).
285         /// </summary>
286         /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
287         /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
288         /// <returns>Индекс искомой связи.</returns>
289         [MethodImpl(MethodImplOptions.AggressiveInlining)]
290         public abstract TLink Search(TLink source, TLink target);
291
292         /// <summary>
293         /// <para>
294         /// Searches the core using the specified root.
295         /// </para>
296         /// <para></para>
297         /// </summary>
298         /// <param name="root">
299         /// <para>The root.</para>
300         /// <para></para>
301         /// </param>
302         /// <param name="key">
303         /// <para>The key.</para>
304         /// <para></para>
305         /// </param>
306         /// <returns>
307         /// <para>The zero.</para>
308         /// <para></para>
309         /// </returns>
310         [MethodImpl(MethodImplOptions.AggressiveInlining)]
311         protected TLink SearchCore(TLink root, TLink key)
312         {
313             while (!EqualToZero(root))
314             {
315                 var rootKey = GetKeyPartValue(root);
316                 if (LessThan(key, rootKey)) // node.Key < root.Key
317                 {
318                     root = GetLeftOrDefault(root);
319                 }
320                 else if (GreaterThan(key, rootKey)) // node.Key > root.Key
321                 {
322                     root = GetRightOrDefault(root);
323                 }
324                 else // node.Key == root.Key
325                 {
326                     return root;
327                 }
328             }
329             return Zero;
330         }
331
332         // TODO: Return indices range instead of references count
333         /// <summary>
334         /// <para>
335         /// Counts the usages using the specified link.
336         /// </para>
337         /// <para></para>
338         /// </summary>
339         /// <param name="link">
340         /// <para>The link.</para>
341         /// <para></para>
342         /// </param>
343         /// <returns>
344         /// <para>The link</para>
345         /// <para></para>
346         /// </returns>
347         [MethodImpl(MethodImplOptions.AggressiveInlining)]
348         public TLink CountUsages(TLink link) => GetSizeOrZero(GetTreeRoot(link));
349
350         /// <summary>
351         /// <para>
352         /// Eaches the usage using the specified base.
353         /// </para>
354         /// <para></para>
355         /// </summary>
356         /// <param name="@base">
357         /// <para>The base.</para>
358         /// <para></para>
```

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

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

```
369         public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
            ↪  EachUsageCore(@base, GetTreeRoot(@base), handler);
370
371         // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
            ↪  low-level MSIL stack.
372         /// <summary>
373         /// <para>
374         /// Eaches the usage core using the specified base.
375         /// </para>
376         /// <para></para>
377         /// </summary>
378         /// <param name="@base">
379         /// <para>The base.</para>
380         /// <para></para>
381         /// </param>
382         /// <param name="link">
383         /// <para>The link.</para>
384         /// <para></para>
385         /// </param>
386         /// <param name="handler">
387         /// <para>The handler.</para>
388         /// <para></para>
389         /// </param>
390         /// <returns>
391         /// <para>The continue.</para>
392         /// <para></para>
393         /// </returns>
394         [MethodImpl(MethodImplOptions.AggressiveInlining)]
395         private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
396         {
397             var @continue = Continue;
398             if (EqualToZero(link))
399             {
400                 return @continue;
401             }
402             var @break = Break;
403             if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
404             {
405                 return @break;
406             }
407             if (AreEqual(handler(GetLinkValues(link)), @break))
408             {
409                 return @break;
410             }
411             if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
412             {
413                 return @break;
414             }
415             return @continue;
416         }
417
418         /// <summary>
419         /// <para>
420         /// Prints the node value using the specified node.
421         /// </para>
422         /// <para></para>
423         /// </summary>
424         /// <param name="node">
425         /// <para>The node.</para>
426         /// <para></para>
427         /// </param>
428         /// <param name="sb">
429         /// <para>The sb.</para>
430         /// <para></para>
431         /// </param>
432         [MethodImpl(MethodImplOptions.AggressiveInlining)]
433         protected override void PrintNodeValue(TLink node, StringBuilder sb)
434         {
435             ref var link = ref GetLinkDataPartReference(node);
436             sb.Append(' ');
437             sb.Append(link.Source);
438             sb.Append('-');
439             sb.Append('>');
440             sb.Append(link.Target);
441         }
442     }
443 }
```

1.37 ./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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪ UncheckedConverter<TLink, long>.Default;
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly byte* _linksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        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)]
```

```
76          public InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants, byte*
            ↪ linksDataParts, byte* linksIndexParts)
77          {
78              _linksDataParts = linksDataParts;
79              _linksIndexParts = linksIndexParts;
80              Break = constants.Break;
81              Continue = constants.Continue;
82          }
83
84          /// <summary>
85          /// <para>
86          /// Gets the link data part reference using the specified link.
87          /// </para>
88          /// <para></para>
89          /// </summary>
90          /// <param name="link">
91          /// <para>The link.</para>
92          /// <para></para>
93          /// </param>
94          /// <returns>
95          /// <para>A ref raw link data part of t link</para>
96          /// <para></para>
97          /// </returns>
98          [MethodImpl(MethodImplOptions.AggressiveInlining)]
99          protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
            ↪ AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (RawLinkDataPart<TLink>.SizeInBytes
            ↪ * _addressToInt64Converter.Convert(link)));
100
101         /// <summary>
102         /// <para>
103         /// Gets the link index part reference using the specified link.
104         /// </para>
105         /// <para></para>
106         /// </summary>
107         /// <param name="link">
108         /// <para>The link.</para>
109         /// <para></para>
110         /// </param>
111         /// <returns>
112         /// <para>A ref raw link index part of t link</para>
113         /// <para></para>
114         /// </returns>
115         [MethodImpl(MethodImplOptions.AggressiveInlining)]
116         protected virtual ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
            ↪ ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
            ↪ (RawLinkIndexPart<TLink>.SizeInBytes * _addressToInt64Converter.Convert(link)));
117
118         /// <summary>
119         /// <para>
120         /// Gets the first using the specified head.
121         /// </para>
122         /// <para></para>
123         /// </summary>
124         /// <param name="head">
125         /// <para>The head.</para>
126         /// <para></para>
127         /// </param>
128         /// <returns>
129         /// <para>The link</para>
130         /// <para></para>
131         /// </returns>
132         [MethodImpl(MethodImplOptions.AggressiveInlining)]
133         protected override TLink GetFirst(TLink head) =>
            ↪ GetLinkIndexPartReference(head).RootAsSource;
134
135         /// <summary>
136         /// <para>
137         /// Gets the last using the specified head.
138         /// </para>
139         /// <para></para>
140         /// </summary>
141         /// <param name="head">
142         /// <para>The head.</para>
143         /// <para></para>
144         /// </param>
145         /// <returns>
146         /// <para>The link</para>
147         /// <para></para>
```

```
148             /// </returns>
149             [MethodImpl(MethodImplOptions.AggressiveInlining)]
150             protected override TLink GetLast(TLink head)
151             {
152                 var first = GetLinkIndexPartReference(head).RootAsSource;
153                 if (EqualToZero(first))
154                 {
155                     return first;
156                 }
157                 else
158                 {
159                     return GetPrevious(first);
160                 }
161             }
162 
163             /// <summary>
164             /// <para>
165             /// Gets the previous using the specified element.
166             /// </para>
167             /// <para></para>
168             /// </summary>
169             /// <param name="element">
170             /// <para>The element.</para>
171             /// <para></para>
172             /// </param>
173             /// <returns>
174             /// <para>The link</para>
175             /// <para></para>
176             /// </returns>
177             [MethodImpl(MethodImplOptions.AggressiveInlining)]
178             protected override TLink GetPrevious(TLink element) =>
                ↪ GetLinkIndexPartReference(element).LeftAsSource;
179 
180             /// <summary>
181             /// <para>
182             /// Gets the next using the specified element.
183             /// </para>
184             /// <para></para>
185             /// </summary>
186             /// <param name="element">
187             /// <para>The element.</para>
188             /// <para></para>
189             /// </param>
190             /// <returns>
191             /// <para>The link</para>
192             /// <para></para>
193             /// </returns>
194             [MethodImpl(MethodImplOptions.AggressiveInlining)]
195             protected override TLink GetNext(TLink element) =>
                ↪ GetLinkIndexPartReference(element).RightAsSource;
196 
197             /// <summary>
198             /// <para>
199             /// Gets the size using the specified head.
200             /// </para>
201             /// <para></para>
202             /// </summary>
203             /// <param name="head">
204             /// <para>The head.</para>
205             /// <para></para>
206             /// </param>
207             /// <returns>
208             /// <para>The link</para>
209             /// <para></para>
210             /// </returns>
211             [MethodImpl(MethodImplOptions.AggressiveInlining)]
212             protected override TLink GetSize(TLink head) =>
                ↪ GetLinkIndexPartReference(head).SizeAsSource;
213 
214             /// <summary>
215             /// <para>
216             /// Sets the first using the specified head.
217             /// </para>
218             /// <para></para>
219             /// </summary>
220             /// <param name="head">
221             /// <para>The head.</para>
222             /// <para></para>
```

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

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

        /// <summary>
        /// <para>
        /// Counts the usages 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)]
        public TLink CountUsages(TLink head) => GetSize(head);

        /// <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>
        /// 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;

```

```
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 }
```

1.39 ./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>
```

```
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(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 }
```

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

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

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

1.42 ./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>
```

```
    /// </summary>
    /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
    public unsafe class SplitMemoryLinks<TLink> : SplitMemoryLinksBase<TLink>
    {
        /// <summary>
        /// <para>
        /// The create internal source tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
        /// <summary>
        /// <para>
        /// The create external source tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
        /// <summary>
        /// <para>
        /// The create internal target tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
        /// <summary>
        /// <para>
        /// The create external target tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        private byte* _header;
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        private byte* _linksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        private byte* _linksIndexParts;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="SplitMemoryLinks"/> 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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public SplitMemoryLinks(string dataMemory, string indexMemory) : this(new
        ↪  FileMappedResizableDirectMemory(dataMemory), new
        ↪  FileMappedResizableDirectMemory(indexMemory)) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="SplitMemoryLinks"/> instance.
        /// </para>
        /// <para></para>
```

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

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

```
219             _header = _linksIndexParts;
220             if (_useLinkedList)
221             {
222                 InternalSourcesListMethods = new
                    ↪  InternalLinksSourcesLinkedListMethods<TLink>(Constants, _linksDataParts,
                    ↪  _linksIndexParts);
223             }
224             else
225             {
226                 InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
227             }
228             ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
229             InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
230             ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
231             UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_linksDataParts, _header);
232         }
233
234         /// <summary>
235         /// <para>
236         /// Resets the pointers.
237         /// </para>
238         /// <para></para>
239         /// </summary>
240         [MethodImpl(MethodImplOptions.AggressiveInlining)]
241         protected override void ResetPointers()
242         {
243             base.ResetPointers();
244             _linksDataParts = null;
245             _linksIndexParts = null;
246             _header = null;
247         }
248
249         /// <summary>
250         /// <para>
251         /// Gets the header reference.
252         /// </para>
253         /// <para></para>
254         /// </summary>
255         /// <returns>
256         /// <para>A ref links header of t link</para>
257         /// <para></para>
258         /// </returns>
259         [MethodImpl(MethodImplOptions.AggressiveInlining)]
260         protected override ref LinksHeader<TLink> GetHeaderReference() => ref
            ↪  AsRef<LinksHeader<TLink>>(_header);
261
262         /// <summary>
263         /// <para>
264         /// Gets the link data part reference using the specified link index.
265         /// </para>
266         /// <para></para>
267         /// </summary>
268         /// <param name="linkIndex">
269         /// <para>The link index.</para>
270         /// <para></para>
271         /// </param>
272         /// <returns>
273         /// <para>A ref raw link data part of t link</para>
274         /// <para></para>
275         /// </returns>
276         [MethodImpl(MethodImplOptions.AggressiveInlining)]
277         protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
            ↪  => ref AsRef<RawLinkDataPart<TLink>>(_linksDataParts + (LinkDataPartSizeInBytes *
            ↪  ConvertToInt64(linkIndex)));
278
279         /// <summary>
280         /// <para>
281         /// Gets the link index part reference using the specified link index.
282         /// </para>
283         /// <para></para>
284         /// </summary>
285         /// <param name="linkIndex">
286         /// <para>The link index.</para>
287         /// <para></para>
288         /// </param>
289         /// <returns>
290         /// <para>A ref raw link index part of t link</para>
291         /// <para></para>
```

```
292             /// </returns>
293             [MethodImpl(MethodImplOptions.AggressiveInlining)]
294             protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
                ↪  linkIndex) => ref AsRef<RawLinkIndexPart<TLink>>(_linksIndexParts +
                ↪  (LinkIndexPartSizeInBytes * ConvertToInt64(linkIndex)));
295         }
296     }
```

1.43 ./csharp/Platform.Data.Doublets/Memory/Split/Generic/SplitMemoryLinksBase.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Runtime.CompilerServices;
4   using Platform.Disposables;
5   using Platform.Singletons;
6   using Platform.Converters;
7   using Platform.Numbers;
8   using Platform.Memory;
9   using Platform.Data.Exceptions;
10
11  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13  namespace Platform.Data.Doublets.Memory.Split.Generic
14  {
15      /// <summary>
16      /// <para>
17      /// Represents the split memory links base.
18      /// </para>
19      /// <para></para>
20      /// </summary>
21      /// <seealso cref="DisposableBase"/>
22      /// <seealso cref="ILinks{TLink}"/>
23      public abstract class SplitMemoryLinksBase<TLink> : DisposableBase, ILinks<TLink>
24      {
25          /// <summary>
26          /// <para>
27          /// The default.
28          /// </para>
29          /// <para></para>
30          /// </summary>
31          private static readonly EqualityComparer<TLink> _equalityComparer =
                ↪  EqualityComparer<TLink>.Default;
32          /// <summary>
33          /// <para>
34          /// The default.
35          /// </para>
36          /// <para></para>
37          /// </summary>
38          private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
39          /// <summary>
40          /// <para>
41          /// The default.
42          /// </para>
43          /// <para></para>
44          /// </summary>
45          private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
                ↪  UncheckedConverter<TLink, long>.Default;
46          /// <summary>
47          /// <para>
48          /// The default.
49          /// </para>
50          /// <para></para>
51          /// </summary>
52          private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
                ↪  UncheckedConverter<long, TLink>.Default;
53
54          /// <summary>
55          /// <para>
56          /// The zero.
57          /// </para>
58          /// <para></para>
59          /// </summary>
60          private static readonly TLink _zero = default;
61          /// <summary>
62          /// <para>
63          /// The zero.
64          /// </para>
65          /// <para></para>
66          /// </summary>
67          private static readonly TLink _one = Arithmetic.Increment(_zero);
```

```
68
69          /// <summary>Возвращает размер одной связи в байтах.</summary>
70          /// <remarks>
71          /// Используется только во вне класса, не рекомедуется использовать внутри.
72          /// Так как во вне не обязательно будет доступен unsafe C#.
73          /// </remarks>
74          public static readonly long LinkDataPartSizeInBytes = RawLinkDataPart<TLink>.SizeInBytes;
75
76          /// <summary>
77          /// <para>
78          /// The size in bytes.
79          /// </para>
80          /// <para></para>
81          /// </summary>
82          public static readonly long LinkIndexPartSizeInBytes =
            ↪  RawLinkIndexPart<TLink>.SizeInBytes;
83
84          /// <summary>
85          /// <para>
86          /// The size in bytes.
87          /// </para>
88          /// <para></para>
89          /// </summary>
90          public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
91
92          /// <summary>
93          /// <para>
94          /// The default links size step.
95          /// </para>
96          /// <para></para>
97          /// </summary>
98          public static readonly long DefaultLinksSizeStep = 1 * 1024 * 1024;
99
100         /// <summary>
101         /// <para>
102         /// The data memory.
103         /// </para>
104         /// <para></para>
105         /// </summary>
106         protected readonly IResizableDirectMemory _dataMemory;
107         /// <summary>
108         /// <para>
109         /// The index memory.
110         /// </para>
111         /// <para></para>
112         /// </summary>
113         protected readonly IResizableDirectMemory _indexMemory;
114         /// <summary>
115         /// <para>
116         /// The use linked list.
117         /// </para>
118         /// <para></para>
119         /// </summary>
120         protected readonly bool _useLinkedList;
121         /// <summary>
122         /// <para>
123         /// The data memory reservation step in bytes.
124         /// </para>
125         /// <para></para>
126         /// </summary>
127         protected readonly long _dataMemoryReservationStepInBytes;
128         /// <summary>
129         /// <para>
130         /// The index memory reservation step in bytes.
131         /// </para>
132         /// <para></para>
133         /// </summary>
134         protected readonly long _indexMemoryReservationStepInBytes;
135
136         /// <summary>
137         /// <para>
138         /// The internal sources list methods.
139         /// </para>
140         /// <para></para>
141         /// </summary>
142         protected InternalLinksSourcesLinkedListMethods<TLink> InternalSourcesListMethods;
143         /// <summary>
144         /// <para>
145         /// 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>
        /// Gets the constants value.
        /// </para>
        /// <para></para>
        /// </summary>
        public virtual LinksConstants<TLink> Constants
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="SplitMemoryLinksBase"/> 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>
```

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

```
            if (minimumDataReservedCapacity < _dataMemoryReservationStepInBytes)
            {
                minimumDataReservedCapacity = _dataMemoryReservationStepInBytes;
            }
            var minimumIndexReservedCapacity = allocatedLinks * LinkDataPartSizeInBytes;
            if (minimumIndexReservedCapacity < indexMemory.UsedCapacity)
            {
                minimumIndexReservedCapacity = indexMemory.UsedCapacity;
            }
            if (minimumIndexReservedCapacity < _indexMemoryReservationStepInBytes)
            {
                minimumIndexReservedCapacity = _indexMemoryReservationStepInBytes;
            }
            // Check for alignment
            if (minimumDataReservedCapacity % _dataMemoryReservationStepInBytes > 0)
            {
                minimumDataReservedCapacity = ((minimumDataReservedCapacity /
                ↪  _dataMemoryReservationStepInBytes) * _dataMemoryReservationStepInBytes) +
                ↪  _dataMemoryReservationStepInBytes;
            }
            if (minimumIndexReservedCapacity % _indexMemoryReservationStepInBytes > 0)
            {
                minimumIndexReservedCapacity = ((minimumIndexReservedCapacity /
                ↪  _indexMemoryReservationStepInBytes) * _indexMemoryReservationStepInBytes) +
                ↪  _indexMemoryReservationStepInBytes;
            }
            if (dataMemory.ReservedCapacity != minimumDataReservedCapacity)
            {
                dataMemory.ReservedCapacity = minimumDataReservedCapacity;
            }
            if (indexMemory.ReservedCapacity != minimumIndexReservedCapacity)
            {
                indexMemory.ReservedCapacity = minimumIndexReservedCapacity;
            }
            SetPointers(dataMemory, indexMemory);
            header = ref GetHeaderReference();
            // Ensure correctness _memory.UsedCapacity over _header->AllocatedLinks
            // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
            dataMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
            ↪  LinkDataPartSizeInBytes) + LinkDataPartSizeInBytes; // First link is read only
            ↪  zero link.
            indexMemory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) *
            ↪  LinkIndexPartSizeInBytes) + LinkHeaderSizeInBytes;
            // Ensure correctness _memory.ReservedLinks over _header->ReservedCapacity
            // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
            header.ReservedLinks = ConvertToAddress((dataMemory.ReservedCapacity -
            ↪  LinkDataPartSizeInBytes) / LinkDataPartSizeInBytes);
        }

        /// <summary>
        /// <para>
        /// Counts the restrictions.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <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 Count(IList<TLink> restrictions)
        {
            // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
            if (restrictions.Count == 0)
            {
                return Total;
            }
            var constants = Constants;
            var any = constants.Any;
            var index = restrictions[constants.IndexPart];
```

```
364                 if (restrictions.Count == 1)
365                 {
366                     if (AreEqual(index, any))
367                     {
368                         return Total;
369                     }
370                     return Exists(index) ? GetOne() : GetZero();
371                 }
372                 if (restrictions.Count == 2)
373                 {
374                     var value = restrictions[1];
375                     if (AreEqual(index, any))
376                     {
377                         if (AreEqual(value, any))
378                         {
379                             return Total; // Any - как отсутствие ограничения
380                         }
381                         var externalReferencesRange = constants.ExternalReferencesRange;
382                         if (externalReferencesRange.HasValue &&
                            ↪  externalReferencesRange.Value.Contains(value))
383                         {
384                             return Add(ExternalSourcesTreeMethods.CountUsages(value),
                                ↪  ExternalTargetsTreeMethods.CountUsages(value));
385                         }
386                         else
387                         {
388                             if (_useLinkedList)
389                             {
390                                 return Add(InternalSourcesListMethods.CountUsages(value),
                                    ↪  InternalTargetsTreeMethods.CountUsages(value));
391                             }
392                             else
393                             {
394                                 return Add(InternalSourcesTreeMethods.CountUsages(value),
                                    ↪  InternalTargetsTreeMethods.CountUsages(value));
395                             }
396                         }
397                     }
398                     else
399                     {
400                         if (!Exists(index))
401                         {
402                             return GetZero();
403                         }
404                         if (AreEqual(value, any))
405                         {
406                             return GetOne();
407                         }
408                         ref var storedLinkValue = ref GetLinkDataPartReference(index);
409                         if (AreEqual(storedLinkValue.Source, value) ||
                            ↪  AreEqual(storedLinkValue.Target, value))
410                         {
411                             return GetOne();
412                         }
413                         return GetZero();
414                     }
415                 }
416                 if (restrictions.Count == 3)
417                 {
418                     var externalReferencesRange = constants.ExternalReferencesRange;
419                     var source = restrictions[constants.SourcePart];
420                     var target = restrictions[constants.TargetPart];
421                     if (AreEqual(index, any))
422                     {
423                         if (AreEqual(source, any) && AreEqual(target, any))
424                         {
425                             return Total;
426                         }
427                         else if (AreEqual(source, any))
428                         {
429                             if (externalReferencesRange.HasValue &&
                                ↪  externalReferencesRange.Value.Contains(target))
430                             {
431                                 return ExternalTargetsTreeMethods.CountUsages(target);
432                             }
433                             else
434                             {
435                                 return InternalTargetsTreeMethods.CountUsages(target);
```

```
436                         }
437                     }
438                     else if (AreEqual(target, any))
439                     {
440                         if (externalReferencesRange.HasValue &&
                            ↪  externalReferencesRange.Value.Contains(source))
441                         {
442                             return ExternalSourcesTreeMethods.CountUsages(source);
443                         }
444                         else
445                         {
446                             if (_useLinkedList)
447                             {
448                                 return InternalSourcesListMethods.CountUsages(source);
449                             }
450                             else
451                             {
452                                 return InternalSourcesTreeMethods.CountUsages(source);
453                             }
454                         }
455                     }
456                     else //if(source != Any && target != Any)
457                     {
458                         // Эквивалент Exists(source, target) => Count(Any, source, target) > 0
459                         TLink link;
460                         if (externalReferencesRange.HasValue)
461                         {
462                             if (externalReferencesRange.Value.Contains(source) &&
                                ↪  externalReferencesRange.Value.Contains(target))
463                             {
464                                 link = ExternalSourcesTreeMethods.Search(source, target);
465                             }
466                             else if (externalReferencesRange.Value.Contains(source))
467                             {
468                                 link = InternalTargetsTreeMethods.Search(source, target);
469                             }
470                             else if (externalReferencesRange.Value.Contains(target))
471                             {
472                                 if (_useLinkedList)
473                                 {
474                                     link = ExternalSourcesTreeMethods.Search(source, target);
475                                 }
476                                 else
477                                 {
478                                     link = InternalSourcesTreeMethods.Search(source, target);
479                                 }
480                             }
481                             else
482                             {
483                                 if (_useLinkedList ||
                                    ↪  GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
                                    ↪  InternalTargetsTreeMethods.CountUsages(target)))
484                                 {
485                                     link = InternalTargetsTreeMethods.Search(source, target);
486                                 }
487                                 else
488                                 {
489                                     link = InternalSourcesTreeMethods.Search(source, target);
490                                 }
491                             }
492                         }
493                         else
494                         {
495                             if (_useLinkedList ||
                                ↪  GreaterThan(InternalSourcesTreeMethods.CountUsages(source),
                                ↪  InternalTargetsTreeMethods.CountUsages(target)))
496                             {
497                                 link = InternalTargetsTreeMethods.Search(source, target);
498                             }
499                             else
500                             {
501                                 link = InternalSourcesTreeMethods.Search(source, target);
502                             }
503                         }
504                         return AreEqual(link, constants.Null) ? GetZero() : GetOne();
505                     }
506                 }
507                 else
```

```
508                 {
509                     if (!Exists(index))
510                     {
511                         return GetZero();
512                     }
513                     if (AreEqual(source, any) && AreEqual(target, any))
514                     {
515                         return GetOne();
516                     }
517                     ref var storedLinkValue = ref GetLinkDataPartReference(index);
518                     if (!AreEqual(source, any) && !AreEqual(target, any))
519                     {
520                         if (AreEqual(storedLinkValue.Source, source) &&
                            ↪  AreEqual(storedLinkValue.Target, target))
521                         {
522                             return GetOne();
523                         }
524                         return GetZero();
525                     }
526                     var value = default(TLink);
527                     if (AreEqual(source, any))
528                     {
529                         value = target;
530                     }
531                     if (AreEqual(target, any))
532                     {
533                         value = source;
534                     }
535                     if (AreEqual(storedLinkValue.Source, value) ||
                        ↪  AreEqual(storedLinkValue.Target, value))
536                     {
537                         return GetOne();
538                     }
539                     return GetZero();
540                 }
541             }
542             throw new NotSupportedException("Другие размеры и способы ограничений не
                ↪  поддерживаются.");
543         }
544 
545         /// <summary>
546         /// <para>
547         /// Eaches the handler.
548         /// </para>
549         /// <para></para>
550         /// </summary>
551         /// <param name="handler">
552         /// <para>The handler.</para>
553         /// <para></para>
554         /// </param>
555         /// <param name="restrictions">
556         /// <para>The restrictions.</para>
557         /// <para></para>
558         /// </param>
559         /// <exception cref="NotSupportedException">
560         /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
561         /// <para></para>
562         /// </exception>
563         /// <returns>
564         /// <para>The link</para>
565         /// <para></para>
566         /// </returns>
567         [MethodImpl(MethodImplOptions.AggressiveInlining)]
568         public virtual TLink Each(Func<IList<TLink>, TLink> handler, IList<TLink> restrictions)
569         {
570             var constants = Constants;
571             var @break = constants.Break;
572             if (restrictions.Count == 0)
573             {
574                 for (var link = GetOne(); LessOrEqualThan(link,
                    ↪  GetHeaderReference().AllocatedLinks); link = Increment(link))
575                 {
576                     if (Exists(link) && AreEqual(handler(GetLinkStruct(link)), @break))
577                     {
578                         return @break;
579                     }
580                 }
581                 return @break;
```

```
582                 }
583                 var @continue = constants.Continue;
584                 var any = constants.Any;
585                 var index = restrictions[constants.IndexPart];
586                 if (restrictions.Count == 1)
587                 {
588                     if (AreEqual(index, any))
589                     {
590                         return Each(handler, Array.Empty<TLink>());
591                     }
592                     if (!Exists(index))
593                     {
594                         return @continue;
595                     }
596                     return handler(GetLinkStruct(index));
597                 }
598                 if (restrictions.Count == 2)
599                 {
600                     var value = restrictions[1];
601                     if (AreEqual(index, any))
602                     {
603                         if (AreEqual(value, any))
604                         {
605                             return Each(handler, Array.Empty<TLink>());
606                         }
607                         if (AreEqual(Each(handler, new Link<TLink>(index, value, any)), @break))
608                         {
609                             return @break;
610                         }
611                         return Each(handler, new Link<TLink>(index, any, value));
612                     }
613                     else
614                     {
615                         if (!Exists(index))
616                         {
617                             return @continue;
618                         }
619                         if (AreEqual(value, any))
620                         {
621                             return handler(GetLinkStruct(index));
622                         }
623                         ref var storedLinkValue = ref GetLinkDataPartReference(index);
624                         if (AreEqual(storedLinkValue.Source, value) ||
625                             AreEqual(storedLinkValue.Target, value))
626                         {
627                             return handler(GetLinkStruct(index));
628                         }
629                         return @continue;
630                     }
631                 }
632                 if (restrictions.Count == 3)
633                 {
634                     var externalReferencesRange = constants.ExternalReferencesRange;
635                     var source = restrictions[constants.SourcePart];
636                     var target = restrictions[constants.TargetPart];
637                     if (AreEqual(index, any))
638                     {
639                         if (AreEqual(source, any) && AreEqual(target, any))
640                         {
641                             return Each(handler, Array.Empty<TLink>());
642                         }
643                         else if (AreEqual(source, any))
644                         {
645                             if (externalReferencesRange.HasValue &&
                                ↪  externalReferencesRange.Value.Contains(target))
646                             {
647                                 return ExternalTargetsTreeMethods.EachUsage(target, handler);
648                             }
649                             else
650                             {
651                                 return InternalTargetsTreeMethods.EachUsage(target, handler);
652                             }
653                         }
654                         else if (AreEqual(target, any))
655                         {
656                             if (externalReferencesRange.HasValue &&
                                ↪  externalReferencesRange.Value.Contains(source))
657                             {
```

```
                        return ExternalSourcesTreeMethods.EachUsage(source, handler);
                    }
                    else
                    {
                        if (_useLinkedList)
                        {
                            return InternalSourcesListMethods.EachUsage(source, handler);
                        }
                        else
                        {
                            return InternalSourcesTreeMethods.EachUsage(source, handler);
                        }
                    }
                }
                else //if(source != Any && target != Any)
                {
                    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) ? @continue :
                    ↪  handler(GetLinkStruct(link));
                }
            }
            else
            {
                if (!Exists(index))
                {
                    return @continue;
                }
                if (AreEqual(source, any) && AreEqual(target, any))
                {
```

```
730                         return handler(GetLinkStruct(index));
731                     }
732                     ref var storedLinkValue = ref GetLinkDataPartReference(index);
733                     if (!AreEqual(source, any) && !AreEqual(target, any))
734                     {
735                         if (AreEqual(storedLinkValue.Source, source) &&
736                             AreEqual(storedLinkValue.Target, target))
737                         {
738                             return handler(GetLinkStruct(index));
739                         }
740                         return @continue;
741                     }
742                     var value = default(TLink);
743                     if (AreEqual(source, any))
744                     {
745                         value = target;
746                     }
747                     if (AreEqual(target, any))
748                     {
749                         value = source;
750                     }
751                     if (AreEqual(storedLinkValue.Source, value) ||
752                         AreEqual(storedLinkValue.Target, value))
753                     {
754                         return handler(GetLinkStruct(index));
755                     }
756                     return @continue;
757                 }
758             }
759             throw new NotSupportedException("Другие размеры и способы ограничений не
                ↪ поддерживаются.");
760         }
761 
762         /// <remarks>
763         /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
                ↪ в другом месте (но не в менеджере памяти, а в логике Links)
764         /// </remarks>
765         [MethodImpl(MethodImplOptions.AggressiveInlining)]
766         public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
767         {
768             var constants = Constants;
769             var @null = constants.Null;
770             var externalReferencesRange = constants.ExternalReferencesRange;
771             var linkIndex = restrictions[constants.IndexPart];
772             ref var link = ref GetLinkDataPartReference(linkIndex);
773             var source = link.Source;
774             var target = link.Target;
775             ref var header = ref GetHeaderReference();
776             ref var rootAsSource = ref header.RootAsSource;
777             ref var rootAsTarget = ref header.RootAsTarget;
778             // Будет корректно работать только в том случае, если пространство выделенной связи
                ↪ предварительно заполнено нулями
779             if (!AreEqual(source, @null))
780             {
781                 if (externalReferencesRange.HasValue &&
                    ↪ externalReferencesRange.Value.Contains(source))
782                 {
783                     ExternalSourcesTreeMethods.Detach(ref rootAsSource, linkIndex);
784                 }
785                 else
786                 {
787                     if (_useLinkedList)
788                     {
789                         InternalSourcesListMethods.Detach(source, linkIndex);
790                     }
791                     else
792                     {
793                         InternalSourcesTreeMethods.Detach(ref
                            ↪ GetLinkIndexPartReference(source).RootAsSource, linkIndex);
794                     }
795                 }
796             }
797             if (!AreEqual(target, @null))
798             {
799                 if (externalReferencesRange.HasValue &&
                    ↪ externalReferencesRange.Value.Contains(target))
800                 {
801                     ExternalTargetsTreeMethods.Detach(ref rootAsTarget, linkIndex);
```

```
802                 }
803                 else
804                 {
805                     InternalTargetsTreeMethods.Detach(ref
                        ↪  GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
806                 }
807             }
808             source = link.Source = substitution[constants.SourcePart];
809             target = link.Target = substitution[constants.TargetPart];
810             if (!AreEqual(source, @null))
811             {
812                 if (externalReferencesRange.HasValue &&
                    ↪  externalReferencesRange.Value.Contains(source))
813                 {
814                     ExternalSourcesTreeMethods.Attach(ref rootAsSource, linkIndex);
815                 }
816                 else
817                 {
818                     if (_useLinkedList)
819                     {
820                         InternalSourcesListMethods.AttachAsLast(source, linkIndex);
821                     }
822                     else
823                     {
824                         InternalSourcesTreeMethods.Attach(ref
                            ↪  GetLinkIndexPartReference(source).RootAsSource, linkIndex);
825                     }
826                 }
827             }
828             if (!AreEqual(target, @null))
829             {
830                 if (externalReferencesRange.HasValue &&
                    ↪  externalReferencesRange.Value.Contains(target))
831                 {
832                     ExternalTargetsTreeMethods.Attach(ref rootAsTarget, linkIndex);
833                 }
834                 else
835                 {
836                     InternalTargetsTreeMethods.Attach(ref
                        ↪  GetLinkIndexPartReference(target).RootAsTarget, linkIndex);
837                 }
838             }
839             return linkIndex;
840         }
841 
842         /// <remarks>
843         /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
                ↪  пространство
844         /// </remarks>
845         [MethodImpl(MethodImplOptions.AggressiveInlining)]
846         public virtual TLink Create(IList<TLink> restrictions)
847         {
848             ref var header = ref GetHeaderReference();
849             var freeLink = header.FirstFreeLink;
850             if (!AreEqual(freeLink, Constants.Null))
851             {
852                 UnusedLinksListMethods.Detach(freeLink);
853             }
854             else
855             {
856                 var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
857                 if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
858                 {
859                     throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
860                 }
861                 if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
862                 {
863                     _dataMemory.ReservedCapacity += _dataMemoryReservationStepInBytes;
864                     _indexMemory.ReservedCapacity += _indexMemoryReservationStepInBytes;
865                     SetPointers(_dataMemory, _indexMemory);
866                     header = ref GetHeaderReference();
867                     header.ReservedLinks = ConvertToAddress(_dataMemory.ReservedCapacity /
                        ↪  LinkDataPartSizeInBytes);
868                 }
869                 freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
870                 _dataMemory.UsedCapacity += LinkDataPartSizeInBytes;
871                 _indexMemory.UsedCapacity += LinkIndexPartSizeInBytes;
872             }
```

```
873             return freeLink;
874         }
875 
876         /// <summary>
877         /// <para>
878         /// Deletes the restrictions.
879         /// </para>
880         /// <para></para>
881         /// </summary>
882         /// <param name="restrictions">
883         /// <para>The restrictions.</para>
884         /// <para></para>
885         /// </param>
886         [MethodImpl(MethodImplOptions.AggressiveInlining)]
887         public virtual void Delete(IList<TLink> restrictions)
888         {
889             ref var header = ref GetHeaderReference();
890             var link = restrictions[Constants.IndexPart];
891             if (LessThan(link, header.AllocatedLinks))
892             {
893                 UnusedLinksListMethods.AttachAsFirst(link);
894             }
895             else if (AreEqual(link, header.AllocatedLinks))
896             {
897                 header.AllocatedLinks = Decrement(header.AllocatedLinks);
898                 _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
899                 _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
900                 // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
                    ↪  пока не дойдём до первой существующей связи
901                 // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
902                 while (GreaterThan(header.AllocatedLinks, GetZero()) &&
                    ↪  IsUnusedLink(header.AllocatedLinks))
903                 {
904                     UnusedLinksListMethods.Detach(header.AllocatedLinks);
905                     header.AllocatedLinks = Decrement(header.AllocatedLinks);
906                     _dataMemory.UsedCapacity -= LinkDataPartSizeInBytes;
907                     _indexMemory.UsedCapacity -= LinkIndexPartSizeInBytes;
908                 }
909             }
910         }
911 
912         /// <summary>
913         /// <para>
914         /// Gets the link struct using the specified link index.
915         /// </para>
916         /// <para></para>
917         /// </summary>
918         /// <param name="linkIndex">
919         /// <para>The link index.</para>
920         /// <para></para>
921         /// </param>
922         /// <returns>
923         /// <para>A list of t link</para>
924         /// <para></para>
925         /// </returns>
926         [MethodImpl(MethodImplOptions.AggressiveInlining)]
927         public IList<TLink> GetLinkStruct(TLink linkIndex)
928         {
929             ref var link = ref GetLinkDataPartReference(linkIndex);
930             return new Link<TLink>(linkIndex, link.Source, link.Target);
931         }
932 
933         /// <remarks>
934         /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
            ↪  адрес реально поменялся
935         ///
936         /// Указатель this.links может быть в том же месте,
937         /// так как 0-я связь не используется и имеет такой же размер как Header,
938         /// поэтому header размещается в том же месте, что и 0-я связь
939         /// </remarks>
940         [MethodImpl(MethodImplOptions.AggressiveInlining)]
941         protected abstract void SetPointers(IResizableDirectMemory dataMemory,
            ↪  IResizableDirectMemory indexMemory);
942 
943         /// <summary>
944         /// <para>
945         /// Resets the pointers.
946         /// </para>
```

```
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual void ResetPointers()
        {
            InternalSourcesListMethods = null;
            InternalSourcesTreeMethods = null;
            ExternalSourcesTreeMethods = null;
            InternalTargetsTreeMethods = null;
            ExternalTargetsTreeMethods = null;
            UnusedLinksListMethods = 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 abstract ref LinksHeader<TLink> GetHeaderReference();

        /// <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 abstract ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink 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 abstract ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
        ↪  linkIndex);

        /// <summary>
        /// <para>
        /// Determines whether this instance exists.
        /// </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)]
        protected virtual bool Exists(TLink link)
            => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
```

```
            && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
            && !IsUnusedLink(link);

        /// <summary>
        /// <para>
        /// Determines whether this instance is unused link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="linkIndex">
        /// <para>The link index.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual bool IsUnusedLink(TLink linkIndex)
        {
            if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
            ↪  is not needed
            {
                // TODO: Reduce access to memory in different location (should be enough to use
                ↪  just linkIndexPart)
                ref var linkDataPart = ref GetLinkDataPartReference(linkIndex);
                ref var linkIndexPart = ref GetLinkIndexPartReference(linkIndex);
                return AreEqual(linkIndexPart.SizeAsTarget, default) &&
                ↪  !AreEqual(linkDataPart.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>
```

```
1099            /// <para></para>
1100            /// </returns>
1101            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1102            protected virtual bool AreEqual(TLink first, TLink second) =>
                ↪ _equalityComparer.Equals(first, second);
1103
1104            /// <summary>
1105            /// <para>
1106            /// Determines whether this instance less than.
1107            /// </para>
1108            /// <para></para>
1109            /// </summary>
1110            /// <param name="first">
1111            /// <para>The first.</para>
1112            /// <para></para>
1113            /// </param>
1114            /// <param name="second">
1115            /// <para>The second.</para>
1116            /// <para></para>
1117            /// </param>
1118            /// <returns>
1119            /// <para>The bool</para>
1120            /// <para></para>
1121            /// </returns>
1122            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1123            protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
                ↪ second) < 0;
1124
1125            /// <summary>
1126            /// <para>
1127            /// Determines whether this instance less or equal than.
1128            /// </para>
1129            /// <para></para>
1130            /// </summary>
1131            /// <param name="first">
1132            /// <para>The first.</para>
1133            /// <para></para>
1134            /// </param>
1135            /// <param name="second">
1136            /// <para>The second.</para>
1137            /// <para></para>
1138            /// </param>
1139            /// <returns>
1140            /// <para>The bool</para>
1141            /// <para></para>
1142            /// </returns>
1143            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1144            protected virtual bool LessOrEqualThan(TLink first, TLink second) =>
                ↪ _comparer.Compare(first, second) <= 0;
1145
1146            /// <summary>
1147            /// <para>
1148            /// Determines whether this instance greater than.
1149            /// </para>
1150            /// <para></para>
1151            /// </summary>
1152            /// <param name="first">
1153            /// <para>The first.</para>
1154            /// <para></para>
1155            /// </param>
1156            /// <param name="second">
1157            /// <para>The second.</para>
1158            /// <para></para>
1159            /// </param>
1160            /// <returns>
1161            /// <para>The bool</para>
1162            /// <para></para>
1163            /// </returns>
1164            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1165            protected virtual bool GreaterThan(TLink first, TLink second) =>
                ↪ _comparer.Compare(first, second) > 0;
1166
1167            /// <summary>
1168            /// <para>
1169            /// Determines whether this instance greater or equal than.
1170            /// </para>
1171            /// <para></para>
1172            /// </summary>
```

```
1173            /// <param name="first">
1174            /// <para>The first.</para>
1175            /// <para></para>
1176            /// </param>
1177            /// <param name="second">
1178            /// <para>The second.</para>
1179            /// <para></para>
1180            /// </param>
1181            /// <returns>
1182            /// <para>The bool</para>
1183            /// <para></para>
1184            /// </returns>
1185            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1186            protected virtual bool GreaterOrEqualThan(TLink first, TLink second) =>
                ↪  _comparer.Compare(first, second) >= 0;
1187
1188            /// <summary>
1189            /// <para>
1190            /// Converts the to int 64 using the specified value.
1191            /// </para>
1192            /// <para></para>
1193            /// </summary>
1194            /// <param name="value">
1195            /// <para>The value.</para>
1196            /// <para></para>
1197            /// </param>
1198            /// <returns>
1199            /// <para>The long</para>
1200            /// <para></para>
1201            /// </returns>
1202            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1203            protected virtual long ConvertToInt64(TLink value) =>
                ↪  _addressToInt64Converter.Convert(value);
1204
1205            /// <summary>
1206            /// <para>
1207            /// Converts the to address using the specified value.
1208            /// </para>
1209            /// <para></para>
1210            /// </summary>
1211            /// <param name="value">
1212            /// <para>The value.</para>
1213            /// <para></para>
1214            /// </param>
1215            /// <returns>
1216            /// <para>The link</para>
1217            /// <para></para>
1218            /// </returns>
1219            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1220            protected virtual TLink ConvertToAddress(long value) =>
                ↪  _int64ToAddressConverter.Convert(value);
1221
1222            /// <summary>
1223            /// <para>
1224            /// Adds the first.
1225            /// </para>
1226            /// <para></para>
1227            /// </summary>
1228            /// <param name="first">
1229            /// <para>The first.</para>
1230            /// <para></para>
1231            /// </param>
1232            /// <param name="second">
1233            /// <para>The second.</para>
1234            /// <para></para>
1235            /// </param>
1236            /// <returns>
1237            /// <para>The link</para>
1238            /// <para></para>
1239            /// </returns>
1240            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1241            protected virtual TLink Add(TLink first, TLink second) => Arithmetic<TLink>.Add(first,
                ↪  second);
1242
1243            /// <summary>
1244            /// <para>
1245            /// Subtracts the first.
1246            /// </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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);

        #region Disposable

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

        /// <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)
            {
                ResetPointers();
                _dataMemory.DisposeIfPossible();
                _indexMemory.DisposeIfPossible();
            }
        }

        #endregion
    }
}
```

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

```
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 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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;

        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly byte* _links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        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)
        {
```

```
60              _links = links;
61              _header = header;
62          }
63
64          /// <summary>
65          /// <para>
66          /// Gets the header reference.
67          /// </para>
68          /// <para></para>
69          /// </summary>
70          /// <returns>
71          /// <para>A ref links header of t link</para>
72          /// <para></para>
73          /// </returns>
74          [MethodImpl(MethodImplOptions.AggressiveInlining)]
75          protected virtual ref LinksHeader<TLink> GetHeaderReference() => ref
            ↪ AsRef<LinksHeader<TLink>>(_header);
76
77          /// <summary>
78          /// <para>
79          /// Gets the link data part reference using the specified link.
80          /// </para>
81          /// <para></para>
82          /// </summary>
83          /// <param name="link">
84          /// <para>The link.</para>
85          /// <para></para>
86          /// </param>
87          /// <returns>
88          /// <para>A ref raw link data part of t link</para>
89          /// <para></para>
90          /// </returns>
91          [MethodImpl(MethodImplOptions.AggressiveInlining)]
92          protected virtual ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) => ref
            ↪ AsRef<RawLinkDataPart<TLink>>(_links + (RawLinkDataPart<TLink>.SizeInBytes *
            ↪ _addressToInt64Converter.Convert(link)));
93
94          /// <summary>
95          /// <para>
96          /// Gets the first.
97          /// </para>
98          /// <para></para>
99          /// </summary>
100         /// <returns>
101         /// <para>The link</para>
102         /// <para></para>
103         /// </returns>
104         [MethodImpl(MethodImplOptions.AggressiveInlining)]
105         protected override TLink GetFirst() => GetHeaderReference().FirstFreeLink;
106
107         /// <summary>
108         /// <para>
109         /// Gets the last.
110         /// </para>
111         /// <para></para>
112         /// </summary>
113         /// <returns>
114         /// <para>The link</para>
115         /// <para></para>
116         /// </returns>
117         [MethodImpl(MethodImplOptions.AggressiveInlining)]
118         protected override TLink GetLast() => GetHeaderReference().LastFreeLink;
119
120         /// <summary>
121         /// <para>
122         /// Gets the previous using the specified element.
123         /// </para>
124         /// <para></para>
125         /// </summary>
126         /// <param name="element">
127         /// <para>The element.</para>
128         /// <para></para>
129         /// </param>
130         /// <returns>
131         /// <para>The link</para>
132         /// <para></para>
133         /// </returns>
134         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
135         protected override TLink GetPrevious(TLink element) =>
            ↪ GetLinkDataPartReference(element).Source;
136
137         /// <summary>
138         /// <para>
139         /// Gets the next using the specified element.
140         /// </para>
141         /// <para></para>
142         /// </summary>
143         /// <param name="element">
144         /// <para>The element.</para>
145         /// <para></para>
146         /// </param>
147         /// <returns>
148         /// <para>The link</para>
149         /// <para></para>
150         /// </returns>
151         [MethodImpl(MethodImplOptions.AggressiveInlining)]
152         protected override TLink GetNext(TLink element) =>
            ↪ GetLinkDataPartReference(element).Target;
153
154         /// <summary>
155         /// <para>
156         /// Gets the size.
157         /// </para>
158         /// <para></para>
159         /// </summary>
160         /// <returns>
161         /// <para>The link</para>
162         /// <para></para>
163         /// </returns>
164         [MethodImpl(MethodImplOptions.AggressiveInlining)]
165         protected override TLink GetSize() => GetHeaderReference().FreeLinks;
166
167         /// <summary>
168         /// <para>
169         /// Sets the first using the specified element.
170         /// </para>
171         /// <para></para>
172         /// </summary>
173         /// <param name="element">
174         /// <para>The element.</para>
175         /// <para></para>
176         /// </param>
177         [MethodImpl(MethodImplOptions.AggressiveInlining)]
178         protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
            ↪ element;
179
180         /// <summary>
181         /// <para>
182         /// Sets the last using the specified element.
183         /// </para>
184         /// <para></para>
185         /// </summary>
186         /// <param name="element">
187         /// <para>The element.</para>
188         /// <para></para>
189         /// </param>
190         [MethodImpl(MethodImplOptions.AggressiveInlining)]
191         protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
            ↪ element;
192
193         /// <summary>
194         /// <para>
195         /// Sets the previous using the specified element.
196         /// </para>
197         /// <para></para>
198         /// </summary>
199         /// <param name="element">
200         /// <para>The element.</para>
201         /// <para></para>
202         /// </param>
203         /// <param name="previous">
204         /// <para>The previous.</para>
205         /// <para></para>
206         /// </param>
207         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
208         protected override void SetPrevious(TLink element, TLink previous) =>
            ↪ GetLinkDataPartReference(element).Source = previous;
209
210         /// <summary>
211         /// <para>
212         /// Sets the next using the specified element.
213         /// </para>
214         /// <para></para>
215         /// </summary>
216         /// <param name="element">
217         /// <para>The element.</para>
218         /// <para></para>
219         /// </param>
220         /// <param name="next">
221         /// <para>The next.</para>
222         /// <para></para>
223         /// </param>
224         [MethodImpl(MethodImplOptions.AggressiveInlining)]
225         protected override void SetNext(TLink element, TLink next) =>
            ↪ GetLinkDataPartReference(element).Target = next;
226
227         /// <summary>
228         /// <para>
229         /// Sets the size using the specified size.
230         /// </para>
231         /// <para></para>
232         /// </summary>
233         /// <param name="size">
234         /// <para>The size.</para>
235         /// <para></para>
236         /// </param>
237         [MethodImpl(MethodImplOptions.AggressiveInlining)]
238         protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
239     }
240 }
```

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

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

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

```
    /// <para>
    /// The raw link index part.
    /// </para>
    /// <para></para>
    /// </summary>
    public struct RawLinkIndexPart<TLink> : IEquatable<RawLinkIndexPart<TLink>>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

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

        /// <summary>
        /// <para>
        /// The root as source.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink RootAsSource;
        /// <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 root as target.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink RootAsTarget;
        /// <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 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;

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

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

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

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

```
        /// </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)
        {
            ref var firstLink = ref LinksDataParts[first];
            ref var secondLink = ref LinksDataParts[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(TLink first, TLink second)
        {
            ref var firstLink = ref LinksDataParts[first];
            ref var secondLink = ref LinksDataParts[second];
            return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
        }
    }
}
```

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

```
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 size balanced tree methods base.
    /// </para>
```

```
    /// <para></para>
    /// </summary>
    /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class UInt32ExternalLinksSizeBalancedTreeMethodsBase :
    ↪  ExternalLinksSizeBalancedTreeMethodsBase<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="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
        ↪  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 UInt32ExternalLinksSizeBalancedTreeMethodsBase(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 == 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>
```

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

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

```
        [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)
        {
            ref var firstLink = ref LinksDataParts[first];
            ref var secondLink = ref LinksDataParts[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(TLink first, TLink second)
        {
            ref var firstLink = ref LinksDataParts[first];
            ref var secondLink = ref LinksDataParts[second];
            return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪ secondLink.Source, secondLink.Target);
        }
    }
}
```

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

```
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 sources recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods :
    ↪ UInt32ExternalLinksRecursionlessSizeBalancedTreeMethodsBase
    {
```

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

```
        /// <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.
        /// </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>
```

```
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 }
```

1.50 ./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">
```

```
        /// <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 UInt32ExternalLinksSourcesSizeBalancedTreeMethods(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>
```

```
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 }
```

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

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

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

```
105             /// <para></para>
106             /// </returns>
107             [MethodImpl(MethodImplOptions.AggressiveInlining)]
108             protected override TLink GetRight(TLink node) => LinksIndexParts[node].RightAsTarget;
109
110             /// <summary>
111             /// <para>
112             /// Sets the left using the specified node.
113             /// </para>
114             /// <para></para>
115             /// </summary>
116             /// <param name="node">
117             /// <para>The node.</para>
118             /// <para></para>
119             /// </param>
120             /// <param name="left">
121             /// <para>The left.</para>
122             /// <para></para>
123             /// </param>
124             [MethodImpl(MethodImplOptions.AggressiveInlining)]
125             protected override void SetLeft(TLink node, TLink left) =>
                ↪ 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.
181         /// </para>
182         /// <para></para>
183         /// </summary>
184         /// <returns>
185         /// <para>The link</para>
186         /// <para></para>
187         /// </returns>
188         [MethodImpl(MethodImplOptions.AggressiveInlining)]
189         protected override TLink GetTreeRoot() => Header->RootAsTarget;
190 
191         /// <summary>
192         /// <para>
193         /// Gets the base part value using the specified node.
194         /// </para>
195         /// <para></para>
196         /// </summary>
197         /// <param name="node">
198         /// <para>The node.</para>
199         /// <para></para>
200         /// </param>
201         /// <returns>
202         /// <para>The link</para>
203         /// <para></para>
204         /// </returns>
205         [MethodImpl(MethodImplOptions.AggressiveInlining)]
206         protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
207 
208         /// <summary>
209         /// <para>
210         /// Determines whether this instance first is to the left of second.
211         /// </para>
212         /// <para></para>
213         /// </summary>
214         /// <param name="firstSource">
215         /// <para>The first source.</para>
216         /// <para></para>
217         /// </param>
218         /// <param name="firstTarget">
219         /// <para>The first target.</para>
220         /// <para></para>
221         /// </param>
222         /// <param name="secondSource">
223         /// <para>The second source.</para>
224         /// <para></para>
225         /// </param>
226         /// <param name="secondTarget">
227         /// <para>The second target.</para>
228         /// <para></para>
229         /// </param>
230         /// <returns>
231         /// <para>The bool</para>
232         /// <para></para>
233         /// </returns>
234         [MethodImpl(MethodImplOptions.AggressiveInlining)]
235         protected override bool 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 }
```

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

```
1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3 
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5 
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 external links targets size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32ExternalLinksSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32ExternalLinksTargetsSizeBalancedTreeMethods :
        ↪ UInt32ExternalLinksSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see cref="UInt32ExternalLinksTargetsSizeBalancedTreeMethods"/>
            ↪ 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 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         /// <summary>
18         /// <para>
19         /// The links data parts.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         private readonly RawLinkDataPart<TLink>* _linksDataParts;
24         /// <summary>
25         /// <para>
26         /// The links index parts.
27         /// </para>
```

```
        /// <para></para>
        /// </summary>
        private readonly RawLinkIndexPart<TLink>* _linksIndexParts;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32InternalLinksSourcesLinkedListMethods"/> 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 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>
        /// Initializes a new <see
        ↪ cref="UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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
        ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(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);
    }
}
```

1.57 ./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;

```

```
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 }
```

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

```
1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt32;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 internal links targets recursionless size balanced tree methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/>
15     public unsafe class UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods :
        ↪ UInt32InternalLinksRecursionlessSizeBalancedTreeMethodsBase
16     {
17         /// <summary>
18         /// <para>
19         /// Initializes a new <see
            ↪ cref="UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         /// <param name="constants">
24         /// <para>A constants.</para>
25         /// <para></para>
26         /// </param>
27         /// <param name="linksDataParts">
28         /// <para>A links data parts.</para>
29         /// <para></para>
30         /// </param>
31         /// <param name="linksIndexParts">
32         /// <para>A links index parts.</para>
33         /// <para></para>
34         /// </param>
35         /// <param name="header">
36         /// <para>A header.</para>
37         /// <para></para>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public
            ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink>
            ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
            ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
            ↪ linksIndexParts, header) { }
41
42         /// <summary>
43         /// <para>
44         /// Gets the left reference using the specified node.
45         /// </para>
46         /// <para></para>
47         /// </summary>
48         /// <param name="node">
49         /// <para>The node.</para>
50         /// <para></para>
51         /// </param>
52         /// <returns>
53         /// <para>The ref link</para>
54         /// <para></para>
55         /// </returns>
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         protected override ref TLink GetLeftReference(TLink node) => ref
            ↪ LinksIndexParts[node].LeftAsTarget;
58
59         /// <summary>
60         /// <para>
61         /// Gets the right reference using the specified node.
62         /// </para>
```

```
        /// <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 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>
        /// </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>
```

```
21          /// <para></para>
22          /// </summary>
23          /// <param name="constants">
24          /// <para>A constants.</para>
25          /// <para></para>
26          /// </param>
27          /// <param name="linksDataParts">
28          /// <para>A links data parts.</para>
29          /// <para></para>
30          /// </param>
31          /// <param name="linksIndexParts">
32          /// <para>A links index parts.</para>
33          /// <para></para>
34          /// </param>
35          /// <param name="header">
36          /// <para>A header.</para>
37          /// <para></para>
38          /// </param>
39          [MethodImpl(MethodImplOptions.AggressiveInlining)]
40          public UInt32InternalLinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink>
            ↪  constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
            ↪  linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
            ↪  linksIndexParts, header) { }
41
42          /// <summary>
43          /// <para>
44          /// Gets the left reference using the specified node.
45          /// </para>
46          /// <para></para>
47          /// </summary>
48          /// <param name="node">
49          /// <para>The node.</para>
50          /// <para></para>
51          /// </param>
52          /// <returns>
53          /// <para>The ref link</para>
54          /// <para></para>
55          /// </returns>
56          [MethodImpl(MethodImplOptions.AggressiveInlining)]
57          protected override ref TLink GetLeftReference(TLink node) => ref
            ↪  LinksIndexParts[node].LeftAsTarget;
58
59          /// <summary>
60          /// <para>
61          /// Gets the right reference using the specified node.
62          /// </para>
63          /// <para></para>
64          /// </summary>
65          /// <param name="node">
66          /// <para>The node.</para>
67          /// <para></para>
68          /// </param>
69          /// <returns>
70          /// <para>The ref link</para>
71          /// <para></para>
72          /// </returns>
73          [MethodImpl(MethodImplOptions.AggressiveInlining)]
74          protected override ref TLink GetRightReference(TLink node) => ref
            ↪  LinksIndexParts[node].RightAsTarget;
75
76          /// <summary>
77          /// <para>
78          /// Gets the left using the specified node.
79          /// </para>
80          /// <para></para>
81          /// </summary>
82          /// <param name="node">
83          /// <para>The node.</para>
84          /// <para></para>
85          /// </param>
86          /// <returns>
87          /// <para>The link</para>
88          /// <para></para>
89          /// </returns>
90          [MethodImpl(MethodImplOptions.AggressiveInlining)]
91          protected override TLink GetLeft(TLink node) => LinksIndexParts[node].LeftAsTarget;
92
93          /// <summary>
```

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

```
        /// </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;
        }
```

```

        /// <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);
    }
}
```

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

```
using System;
using System.Runtime.CompilerServices;
using Platform.Singletons;
using Platform.Memory;
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 split memory links.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
    public unsafe class UInt32SplitMemoryLinks : SplitMemoryLinksBase<TLink>
    {
        /// <summary>
        /// <para>
        /// The create internal source tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
        /// <summary>
        /// <para>
        /// The create external source tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
        /// <summary>
        /// <para>
        /// The create internal target tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
        /// <summary>
        /// <para>
        /// The create external target tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
```

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

```
129         /// <para></para>
130         /// </param>
131         [MethodImpl(MethodImplOptions.AggressiveInlining)]
132         public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
            ↪ this(dataMemory, indexMemory, memoryReservationStep, constants,
            ↪ IndexTreeType.Default, useLinkedList: true) { }
133
134         /// <summary>
135         /// <para>
136         /// Initializes a new <see cref="UInt32SplitMemoryLinks"/> instance.
137         /// </para>
138         /// <para></para>
139         /// </summary>
140         /// <param name="dataMemory">
141         /// <para>A data memory.</para>
142         /// <para></para>
143         /// </param>
144         /// <param name="indexMemory">
145         /// <para>A index memory.</para>
146         /// <para></para>
147         /// </param>
148         /// <param name="memoryReservationStep">
149         /// <para>A memory reservation step.</para>
150         /// <para></para>
151         /// </param>
152         /// <param name="constants">
153         /// <para>A constants.</para>
154         /// <para></para>
155         /// </param>
156         /// <param name="indexTreeType">
157         /// <para>A index tree type.</para>
158         /// <para></para>
159         /// </param>
160         /// <param name="useLinkedList">
161         /// <para>A use linked list.</para>
162         /// <para></para>
163         /// </param>
164         [MethodImpl(MethodImplOptions.AggressiveInlining)]
165         public UInt32SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪ indexMemory, long memoryReservationStep, LinksConstants<TLink> constants,
            ↪ IndexTreeType indexTreeType, bool useLinkedList) : base(dataMemory, indexMemory,
            ↪ memoryReservationStep, constants, useLinkedList)
166         {
167             if (indexTreeType == IndexTreeType.SizeBalancedTree)
168             {
169                 _createInternalSourceTreeMethods = () => new
                    ↪ UInt32InternalLinksSourcesSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
170                 _createExternalSourceTreeMethods = () => new
                    ↪ UInt32ExternalLinksSourcesSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
171                 _createInternalTargetTreeMethods = () => new
                    ↪ UInt32InternalLinksTargetsSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
172                 _createExternalTargetTreeMethods = () => new
                    ↪ UInt32ExternalLinksTargetsSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
173             }
174             else
175             {
176                 _createInternalSourceTreeMethods = () => new
                    ↪ UInt32InternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
177                 _createExternalSourceTreeMethods = () => new
                    ↪ UInt32ExternalLinksSourcesRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
178                 _createInternalTargetTreeMethods = () => new
                    ↪ UInt32InternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
179                 _createExternalTargetTreeMethods = () => new
                    ↪ UInt32ExternalLinksTargetsRecursionlessSizeBalancedTreeMethods(Constants,
                    ↪ _linksDataParts, _linksIndexParts, _header);
180             }
181             Init(dataMemory, indexMemory);
182         }
183
```

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

```
259         /// </returns>
260         [MethodImpl(MethodImplOptions.AggressiveInlining)]
261         protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink linkIndex)
            ↪  => ref _linksDataParts[linkIndex];
262
263         /// <summary>
264         /// <para>
265         /// Gets the link index part reference using the specified link index.
266         /// </para>
267         /// <para></para>
268         /// </summary>
269         /// <param name="linkIndex">
270         /// <para>The link index.</para>
271         /// <para></para>
272         /// </param>
273         /// <returns>
274         /// <para>A ref raw link index part of t link</para>
275         /// <para></para>
276         /// </returns>
277         [MethodImpl(MethodImplOptions.AggressiveInlining)]
278         protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink
            ↪  linkIndex) => ref _linksIndexParts[linkIndex];
279
280         /// <summary>
281         /// <para>
282         /// Determines whether this instance are equal.
283         /// </para>
284         /// <para></para>
285         /// </summary>
286         /// <param name="first">
287         /// <para>The first.</para>
288         /// <para></para>
289         /// </param>
290         /// <param name="second">
291         /// <para>The second.</para>
292         /// <para></para>
293         /// </param>
294         /// <returns>
295         /// <para>The bool</para>
296         /// <para></para>
297         /// </returns>
298         [MethodImpl(MethodImplOptions.AggressiveInlining)]
299         protected override bool AreEqual(TLink first, TLink second) => first == second;
300
301         /// <summary>
302         /// <para>
303         /// Determines whether this instance less than.
304         /// </para>
305         /// <para></para>
306         /// </summary>
307         /// <param name="first">
308         /// <para>The first.</para>
309         /// <para></para>
310         /// </param>
311         /// <param name="second">
312         /// <para>The second.</para>
313         /// <para></para>
314         /// </param>
315         /// <returns>
316         /// <para>The bool</para>
317         /// <para></para>
318         /// </returns>
319         [MethodImpl(MethodImplOptions.AggressiveInlining)]
320         protected override bool LessThan(TLink first, TLink second) => first < second;
321
322         /// <summary>
323         /// <para>
324         /// Determines whether this instance less or equal than.
325         /// </para>
326         /// <para></para>
327         /// </summary>
328         /// <param name="first">
329         /// <para>The first.</para>
330         /// <para></para>
331         /// </param>
332         /// <param name="second">
333         /// <para>The second.</para>
334         /// <para></para>
```

```
        /// </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>
```

```
413         /// Converts the to int 64 using the specified value.
414         /// </para>
415         /// <para></para>
416         /// </summary>
417         /// <param name="value">
418         /// <para>The value.</para>
419         /// <para></para>
420         /// </param>
421         /// <returns>
422         /// <para>The long</para>
423         /// <para></para>
424         /// </returns>
425         [MethodImpl(MethodImplOptions.AggressiveInlining)]
426         protected override long ConvertToInt64(TLink value) => value;
427
428         /// <summary>
429         /// <para>
430         /// Converts the to address using the specified value.
431         /// </para>
432         /// <para></para>
433         /// </summary>
434         /// <param name="value">
435         /// <para>The value.</para>
436         /// <para></para>
437         /// </param>
438         /// <returns>
439         /// <para>The link</para>
440         /// <para></para>
441         /// </returns>
442         [MethodImpl(MethodImplOptions.AggressiveInlining)]
443         protected override TLink ConvertToAddress(long value) => (TLink)value;
444
445         /// <summary>
446         /// <para>
447         /// Adds the first.
448         /// </para>
449         /// <para></para>
450         /// </summary>
451         /// <param name="first">
452         /// <para>The first.</para>
453         /// <para></para>
454         /// </param>
455         /// <param name="second">
456         /// <para>The second.</para>
457         /// <para></para>
458         /// </param>
459         /// <returns>
460         /// <para>The link</para>
461         /// <para></para>
462         /// </returns>
463         [MethodImpl(MethodImplOptions.AggressiveInlining)]
464         protected override TLink Add(TLink first, TLink second) => first + second;
465
466         /// <summary>
467         /// <para>
468         /// Subtracts the first.
469         /// </para>
470         /// <para></para>
471         /// </summary>
472         /// <param name="first">
473         /// <para>The first.</para>
474         /// <para></para>
475         /// </param>
476         /// <param name="second">
477         /// <para>The second.</para>
478         /// <para></para>
479         /// </param>
480         /// <returns>
481         /// <para>The link</para>
482         /// <para></para>
483         /// </returns>
484         [MethodImpl(MethodImplOptions.AggressiveInlining)]
485         protected override TLink Subtract(TLink first, TLink second) => first - second;
486
487         /// <summary>
488         /// <para>
489         /// Increments the link.
490         /// </para>
```

```
        /// <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 Increment(TLink link) => ++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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink Decrement(TLink link) => --link;
    }
}
```

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

```
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 unused links list methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UnusedLinksListMethods{TLink}"/>
    public unsafe class UInt32UnusedLinksListMethods : UnusedLinksListMethods<TLink>
    {
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly RawLinkDataPart<TLink>* _links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly LinksHeader<TLink>* _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>
```

```
47          [MethodImpl(MethodImplOptions.AggressiveInlining)]
48          public UInt32UnusedLinksListMethods(RawLinkDataPart<TLink>* links, LinksHeader<TLink>*
            ↪ header)
49              : base((byte*)links, (byte*)header)
50          {
51              _links = links;
52              _header = header;
53          }
54
55          /// <summary>
56          /// <para>
57          /// Gets the link data part reference using the specified link.
58          /// </para>
59          /// <para></para>
60          /// </summary>
61          /// <param name="link">
62          /// <para>The link.</para>
63          /// <para></para>
64          /// </param>
65          /// <returns>
66          /// <para>A ref raw link data part of t link</para>
67          /// <para></para>
68          /// </returns>
69          [MethodImpl(MethodImplOptions.AggressiveInlining)]
70          protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
            ↪ ref _links[link];
71
72          /// <summary>
73          /// <para>
74          /// Gets the header reference.
75          /// </para>
76          /// <para></para>
77          /// </summary>
78          /// <returns>
79          /// <para>A ref links header of t link</para>
80          /// <para></para>
81          /// </returns>
82          [MethodImpl(MethodImplOptions.AggressiveInlining)]
83          protected override ref LinksHeader<TLink> GetHeaderReference() => ref *_header;
84      }
85  }
```

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

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

```
        /// </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>
        /// </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>
        /// <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 LinksDataParts[first];
            ref var secondLink = ref LinksDataParts[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(TLink first, TLink second)
        {
            ref var firstLink = ref LinksDataParts[first];
            ref var secondLink = ref LinksDataParts[second];
            return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪  secondLink.Source, secondLink.Target);
        }
    }
}
```

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

```
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 size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ExternalLinksSizeBalancedTreeMethodsBase{TLink}"/>
    /// <seealso cref="ILinksTreeMethods{TLink}"/>
    public unsafe abstract class UInt64ExternalLinksSizeBalancedTreeMethodsBase :
    ↪  ExternalLinksSizeBalancedTreeMethodsBase<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;
```

```
40
41          /// <summary>
42          /// <para>
43          /// Initializes a new <see cref="UInt64ExternalLinksSizeBalancedTreeMethodsBase"/>
            ↪  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 UInt64ExternalLinksSizeBalancedTreeMethodsBase(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 ulong</para>
80          /// <para></para>
81          /// </returns>
82          [MethodImpl(MethodImplOptions.AggressiveInlining)]
83          protected override ulong GetZero() => 0UL;
84
85          /// <summary>
86          /// <para>
87          /// Determines whether this instance equal to zero.
88          /// </para>
89          /// <para></para>
90          /// </summary>
91          /// <param name="value">
92          /// <para>The value.</para>
93          /// <para></para>
94          /// </param>
95          /// <returns>
96          /// <para>The bool</para>
97          /// <para></para>
98          /// </returns>
99          [MethodImpl(MethodImplOptions.AggressiveInlining)]
100         protected override bool EqualToZero(ulong value) => value == 0UL;
101
102         /// <summary>
103         /// <para>
104         /// Determines whether this instance are equal.
105         /// </para>
106         /// <para></para>
107         /// </summary>
108         /// <param name="first">
109         /// <para>The first.</para>
110         /// <para></para>
111         /// </param>
112         /// <param name="second">
113         /// <para>The second.</para>
114         /// <para></para>
```

```
        /// </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>
        /// <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>
        /// <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 LinksDataParts[first];
            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>
```

```
        /// </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>
```

```
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>
38         /// </param>
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public UInt64ExternalLinksSourcesSizeBalancedTreeMethods(LinksConstants<TLink>
           ↪ constants, RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>*
           ↪ linksIndexParts, LinksHeader<TLink>* header) : base(constants, linksDataParts,
           ↪ linksIndexParts, header) { }
41 
42         /// <summary>
43         /// <para>
44         /// Gets the left reference using the specified node.
45         /// </para>
```

```
        /// <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>
```

```
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.
```

```
        /// </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>
```

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

```
        /// </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)]
```

```
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)
```

```
        {
            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>
        /// <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>
```

```
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.
181             /// </para>
182             /// <para></para>
183             /// </summary>
184             /// <returns>
185             /// <para>The link</para>
186             /// <para></para>
187             /// </returns>
188             [MethodImpl(MethodImplOptions.AggressiveInlining)]
189             protected override TLink GetTreeRoot() => Header->RootAsTarget;
190
191             /// <summary>
192             /// <para>
193             /// Gets the base part value using the specified node.
194             /// </para>
195             /// <para></para>
196             /// </summary>
197             /// <param name="node">
198             /// <para>The node.</para>
199             /// <para></para>
200             /// </param>
201             /// <returns>
202             /// <para>The link</para>
203             /// <para></para>
204             /// </returns>
205             [MethodImpl(MethodImplOptions.AggressiveInlining)]
206             protected override TLink GetBasePartValue(TLink node) => LinksDataParts[node].Target;
207
208             /// <summary>
209             /// <para>
210             /// Determines whether this instance first is to the left of second.
211             /// </para>
212             /// <para></para>
213             /// </summary>
214             /// <param name="firstSource">
215             /// <para>The first source.</para>
```

```
        /// <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;
        }
    }
}
```

```
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 recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="InternalLinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe abstract class UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase :
    ↪ InternalLinksRecursionlessSizeBalancedTreeMethodsBase<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="UInt64InternalLinksRecursionlessSizeBalancedTreeMethodsBase"/> 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
        ↪ 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() => OUL;

        /// <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 == OUL;

        /// <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 > OUL;

        /// <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>
        /// <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)]
```

```
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 }
```

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

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

```
        /// </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">
        /// <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>
        /// <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>
```

```
334             /// </summary>
335             /// <param name="first">
336             /// <para>The first.</para>
337             /// <para></para>
338             /// </param>
339             /// <param name="second">
340             /// <para>The second.</para>
341             /// <para></para>
342             /// </param>
343             /// <returns>
344             /// <para>The ulong</para>
345             /// <para></para>
346             /// </returns>
347             [MethodImpl(MethodImplOptions.AggressiveInlining)]
348             protected override ulong Subtract(ulong first, ulong second) => first - second;
349
350             /// <summary>
351             /// <para>
352             /// Gets the link data part reference using the specified link.
353             /// </para>
354             /// <para></para>
355             /// </summary>
356             /// <param name="link">
357             /// <para>The link.</para>
358             /// <para></para>
359             /// </param>
360             /// <returns>
361             /// <para>A ref raw link data part of t link</para>
362             /// <para></para>
363             /// </returns>
364             [MethodImpl(MethodImplOptions.AggressiveInlining)]
365             protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
                ↪  ref LinksDataParts[link];
366
367             /// <summary>
368             /// <para>
369             /// Gets the link index part reference using the specified link.
370             /// </para>
371             /// <para></para>
372             /// </summary>
373             /// <param name="link">
374             /// <para>The link.</para>
375             /// <para></para>
376             /// </param>
377             /// <returns>
378             /// <para>A ref raw link index part of t link</para>
379             /// <para></para>
380             /// </returns>
381             [MethodImpl(MethodImplOptions.AggressiveInlining)]
382             protected override ref RawLinkIndexPart<TLink> GetLinkIndexPartReference(TLink link) =>
                ↪  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 }
```

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

```
1  using System.Runtime.CompilerServices;
2  using TLink = System.UInt64;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.Split.Generic
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 64 internal links sources linked list methods.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="InternalLinksSourcesLinkedListMethods{TLink}"/>
15     public unsafe class UInt64InternalLinksSourcesLinkedListMethods :
       ↪ InternalLinksSourcesLinkedListMethods<TLink>
16     {
17         /// <summary>
18         /// <para>
19         /// The links data parts.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         private readonly RawLinkDataPart<TLink>* _linksDataParts;
24         /// <summary>
25         /// <para>
26         /// The links index parts.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         private readonly RawLinkIndexPart<TLink>* _linksIndexParts;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="UInt64InternalLinksSourcesLinkedListMethods"/> instance.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="constants">
39         /// <para>A constants.</para>
40         /// <para></para>
41         /// </param>
42         /// <param name="linksDataParts">
43         /// <para>A links data parts.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="linksIndexParts">
47         /// <para>A links index parts.</para>
48         /// <para></para>
49         /// </param>
50         [MethodImpl(MethodImplOptions.AggressiveInlining)]
51         public UInt64InternalLinksSourcesLinkedListMethods(LinksConstants<TLink> constants,
           ↪ RawLinkDataPart<TLink>* linksDataParts, RawLinkIndexPart<TLink>* linksIndexParts)
52             : base(constants, (byte*)linksDataParts, (byte*)linksIndexParts)
53         {
54             _linksDataParts = linksDataParts;
55             _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;

        /// <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);
    }
}
```

1.72 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksSourcesSizeBalancedTreeMetho

```
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 size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64InternalLinksSourcesSizeBalancedTreeMethods :
    ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64InternalLinksSourcesSizeBalancedTreeMethods"/>
        ↪ 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 UInt64InternalLinksSourcesSizeBalancedTreeMethods(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>
```

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

1.73 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64InternalLinksTargetsRecursionlessSizeBalanc

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

```
        /// <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;
        }

        /// <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);
    }
}
```

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

```
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 size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64InternalLinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64InternalLinksTargetsSizeBalancedTreeMethods :
    ↪ UInt64InternalLinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64InternalLinksTargetsSizeBalancedTreeMethods"/>
        ↪ 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 UInt64InternalLinksTargetsSizeBalancedTreeMethods(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;

        /// <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>
```

```
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;
```

```
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 split memory links.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="SplitMemoryLinksBase{TLink}"/>
    public unsafe class UInt64SplitMemoryLinks : SplitMemoryLinksBase<TLink>
    {
        /// <summary>
        /// <para>
        /// The create internal source tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createInternalSourceTreeMethods;
        /// <summary>
        /// <para>
        /// The create external source tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createExternalSourceTreeMethods;
        /// <summary>
        /// <para>
        /// The create internal target tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createInternalTargetTreeMethods;
        /// <summary>
        /// <para>
        /// The create external target tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<TLink>> _createExternalTargetTreeMethods;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        private LinksHeader<ulong>* _header;
        /// <summary>
        /// <para>
        /// The links data parts.
        /// </para>
        /// <para></para>
        /// </summary>
        private RawLinkDataPart<ulong>* _linksDataParts;
        /// <summary>
        /// <para>
        /// The links index parts.
        /// </para>
        /// <para></para>
        /// </summary>
        private RawLinkIndexPart<ulong>* _linksIndexParts;

        /// <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>
```

```
84          /// </param>
85          [MethodImpl(MethodImplOptions.AggressiveInlining)]
86          public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪  indexMemory) : this(dataMemory, indexMemory, DefaultLinksSizeStep) { }
87
88          /// <summary>
89          /// <para>
90          /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
91          /// </para>
92          /// <para></para>
93          /// </summary>
94          /// <param name="dataMemory">
95          /// <para>A data memory.</para>
96          /// <para></para>
97          /// </param>
98          /// <param name="indexMemory">
99          /// <para>A index memory.</para>
100         /// <para></para>
101         /// </param>
102         /// <param name="memoryReservationStep">
103         /// <para>A memory reservation step.</para>
104         /// <para></para>
105         /// </param>
106         [MethodImpl(MethodImplOptions.AggressiveInlining)]
107         public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪  indexMemory, long memoryReservationStep) : this(dataMemory, indexMemory,
            ↪  memoryReservationStep, Default<LinksConstants<TLink>>.Instance,
            ↪  IndexTreeType.Default, useLinkedList: true) { }
108
109         /// <summary>
110         /// <para>
111         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
112         /// </para>
113         /// <para></para>
114         /// </summary>
115         /// <param name="dataMemory">
116         /// <para>A data memory.</para>
117         /// <para></para>
118         /// </param>
119         /// <param name="indexMemory">
120         /// <para>A index memory.</para>
121         /// <para></para>
122         /// </param>
123         /// <param name="memoryReservationStep">
124         /// <para>A memory reservation step.</para>
125         /// <para></para>
126         /// </param>
127         /// <param name="constants">
128         /// <para>A constants.</para>
129         /// <para></para>
130         /// </param>
131         [MethodImpl(MethodImplOptions.AggressiveInlining)]
132         public UInt64SplitMemoryLinks(IResizableDirectMemory dataMemory, IResizableDirectMemory
            ↪  indexMemory, long memoryReservationStep, LinksConstants<TLink> constants) :
            ↪  this(dataMemory, indexMemory, memoryReservationStep, constants,
            ↪  IndexTreeType.Default, useLinkedList: true) { }
133
134         /// <summary>
135         /// <para>
136         /// Initializes a new <see cref="UInt64SplitMemoryLinks"/> instance.
137         /// </para>
138         /// <para></para>
139         /// </summary>
140         /// <param name="dataMemory">
141         /// <para>A data memory.</para>
142         /// <para></para>
143         /// </param>
144         /// <param name="indexMemory">
145         /// <para>A index memory.</para>
146         /// <para></para>
147         /// </param>
148         /// <param name="memoryReservationStep">
149         /// <para>A memory reservation step.</para>
150         /// <para></para>
151         /// </param>
152         /// <param name="constants">
153         /// <para>A constants.</para>
```

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

```
            {
                InternalSourcesTreeMethods = _createInternalSourceTreeMethods();
            }
            ExternalSourcesTreeMethods = _createExternalSourceTreeMethods();
            InternalTargetsTreeMethods = _createInternalTargetTreeMethods();
            ExternalTargetsTreeMethods = _createExternalTargetTreeMethods();
            UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_linksDataParts, _header);
        }

        /// <summary>
        /// <para>
        /// Resets the pointers.
        /// </para>
        /// <para></para>
        /// </summary>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void ResetPointers()
        {
            base.ResetPointers();
            _linksDataParts = null;
            _linksIndexParts = 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 *_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(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;
```

```
363
364         /// <summary>
365         /// <para>
366         /// Determines whether this instance greater or equal than.
367         /// </para>
368         /// <para></para>
369         /// </summary>
370         /// <param name="first">
371         /// <para>The first.</para>
372         /// <para></para>
373         /// </param>
374         /// <param name="second">
375         /// <para>The second.</para>
376         /// <para></para>
377         /// </param>
378         /// <returns>
379         /// <para>The bool</para>
380         /// <para></para>
381         /// </returns>
382         [MethodImpl(MethodImplOptions.AggressiveInlining)]
383         protected override bool GreaterOrEqualThan(ulong first, ulong second) => first >= second;
384
385         /// <summary>
386         /// <para>
387         /// Gets the zero.
388         /// </para>
389         /// <para></para>
390         /// </summary>
391         /// <returns>
392         /// <para>The ulong</para>
393         /// <para></para>
394         /// </returns>
395         [MethodImpl(MethodImplOptions.AggressiveInlining)]
396         protected override ulong GetZero() => 0UL;
397
398         /// <summary>
399         /// <para>
400         /// Gets the one.
401         /// </para>
402         /// <para></para>
403         /// </summary>
404         /// <returns>
405         /// <para>The ulong</para>
406         /// <para></para>
407         /// </returns>
408         [MethodImpl(MethodImplOptions.AggressiveInlining)]
409         protected override ulong GetOne() => 1UL;
410
411         /// <summary>
412         /// <para>
413         /// Converts the to int 64 using the specified value.
414         /// </para>
415         /// <para></para>
416         /// </summary>
417         /// <param name="value">
418         /// <para>The value.</para>
419         /// <para></para>
420         /// </param>
421         /// <returns>
422         /// <para>The long</para>
423         /// <para></para>
424         /// </returns>
425         [MethodImpl(MethodImplOptions.AggressiveInlining)]
426         protected override long ConvertToInt64(ulong value) => (long)value;
427
428         /// <summary>
429         /// <para>
430         /// Converts the to address using the specified value.
431         /// </para>
432         /// <para></para>
433         /// </summary>
434         /// <param name="value">
435         /// <para>The value.</para>
436         /// <para></para>
437         /// </param>
438         /// <returns>
439         /// <para>The ulong</para>
440         /// <para></para>
```

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

        /// <summary>
        /// <para>
        /// Decrements the 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)]
```

```
519         protected override ulong Decrement(ulong link) => --link;
520     }
521 }
```

1.76 ./csharp/Platform.Data.Doublets/Memory/Split/Specific/UInt64UnusedLinksListMethods.cs

```
1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.Split.Generic;
3  using TLink = System.UInt64;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.Data.Doublets.Memory.Split.Specific
8  {
9      /// <summary>
10     /// <para>
11     /// Represents the int 64 unused links list methods.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="UnusedLinksListMethods{TLink}"/>
16     public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<TLink>
17     {
18         /// <summary>
19         /// <para>
20         /// The links.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private readonly RawLinkDataPart<ulong>* _links;
25         /// <summary>
26         /// <para>
27         /// The header.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         private readonly LinksHeader<ulong>* _header;
32
33         /// <summary>
34         /// <para>
35         /// Initializes a new <see cref="UInt64UnusedLinksListMethods"/> instance.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         /// <param name="links">
40         /// <para>A links.</para>
41         /// <para></para>
42         /// </param>
43         /// <param name="header">
44         /// <para>A header.</para>
45         /// <para></para>
46         /// </param>
47         [MethodImpl(MethodImplOptions.AggressiveInlining)]
48         public UInt64UnusedLinksListMethods(RawLinkDataPart<ulong>* links, LinksHeader<ulong>*
           ↪ header)
49             : base((byte*)links, (byte*)header)
50         {
51             _links = links;
52             _header = header;
53         }
54
55         /// <summary>
56         /// <para>
57         /// Gets the link data part reference using the specified link.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         /// <param name="link">
62         /// <para>The link.</para>
63         /// <para></para>
64         /// </param>
65         /// <returns>
66         /// <para>A ref raw link data part of t link</para>
67         /// <para></para>
68         /// </returns>
69         [MethodImpl(MethodImplOptions.AggressiveInlining)]
70         protected override ref RawLinkDataPart<TLink> GetLinkDataPartReference(TLink link) =>
           ↪ ref _links[link];
71
72         /// <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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪ UncheckedConverter<TLink, long>.Default;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly UncheckedConverter<TLink, int> _addressToInt32Converter =
        ↪ UncheckedConverter<TLink, int>.Default;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly UncheckedConverter<bool, TLink> _boolToAddressConverter =
        ↪ UncheckedConverter<bool, TLink>.Default;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly UncheckedConverter<TLink, bool> _addressToBoolConverter =
        ↪ UncheckedConverter<TLink, bool>.Default;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly UncheckedConverter<int, TLink> _int32ToAddressConverter =
        ↪ UncheckedConverter<int, TLink>.Default;
```

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

        /// <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>
        /// <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)
        {
            ref var firstLink = ref GetLinkReference(first);
            ref var secondLink = ref GetLinkReference(second);
            return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
            ↪  secondLink.Source, secondLink.Target);
        }

        /// <summary>
        /// <para>
        /// Gets the size value 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 GetSizeValue(TLink value) => Bit<TLink>.PartialRead(value, 5,
        ↪  -5);

        /// <summary>
        /// <para>
        /// Sets the size value using the specified stored value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storedValue">
        /// <para>The stored value.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected virtual void SetSizeValue(ref TLink storedValue, TLink size) => storedValue =
        ↪  Bit<TLink>.PartialWrite(storedValue, size, 5, -5);

        /// <summary>
        /// <para>
        /// Determines whether this instance get left 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 GetLeftIsChildValue(TLink value)
        {
            unchecked
            {
                return _addressToBoolConverter.Convert(Bit<TLink>.PartialRead(value, 4, 1));
```

```
                //return !EqualityComparer.Equals(Bit<TLink>.PartialRead(value, 4, 1), default);
            }
        }

        /// <summary>
        /// <para>
        /// 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;
            }
```

```
437         }
438
439         /// <summary>
440         /// <para>
441         /// Determines whether this instance is child.
442         /// </para>
443         /// <para></para>
444         /// </summary>
445         /// <param name="parent">
446         /// <para>The parent.</para>
447         /// <para></para>
448         /// </param>
449         /// <param name="possibleChild">
450         /// <para>The possible child.</para>
451         /// <para></para>
452         /// </param>
453         /// <returns>
454         /// <para>The bool</para>
455         /// <para></para>
456         /// </returns>
457         [MethodImpl(MethodImplOptions.AggressiveInlining)]
458         protected bool IsChild(TLink parent, TLink possibleChild)
459         {
460             var parentSize = GetSize(parent);
461             var childSize = GetSizeOrZero(possibleChild);
462             return GreaterThanZero(childSize) && LessOrEqualThan(childSize, parentSize);
463         }
464
465         /// <summary>
466         /// <para>
467         /// Gets the balance value using the specified stored value.
468         /// </para>
469         /// <para></para>
470         /// </summary>
471         /// <param name="storedValue">
472         /// <para>The stored value.</para>
473         /// <para></para>
474         /// </param>
475         /// <returns>
476         /// <para>The sbyte</para>
477         /// <para></para>
478         /// </returns>
479         [MethodImpl(MethodImplOptions.AggressiveInlining)]
480         protected virtual sbyte GetBalanceValue(TLink storedValue)
481         {
482             unchecked
483             {
484                 var value = _addressToInt32Converter.Convert(Bit<TLink>.PartialRead(storedValue,
                        ↪  0, 3));
485                 value |= 0xF8 * ((value & 4) >> 2); // if negative, then continue ones to the
                        ↪  end of sbyte
486                 return (sbyte)value;
487             }
488         }
489
490         /// <summary>
491         /// <para>
492         /// Sets the balance value using the specified stored value.
493         /// </para>
494         /// <para></para>
495         /// </summary>
496         /// <param name="storedValue">
497         /// <para>The stored value.</para>
498         /// <para></para>
499         /// </param>
500         /// <param name="value">
501         /// <para>The value.</para>
502         /// <para></para>
503         /// </param>
504         [MethodImpl(MethodImplOptions.AggressiveInlining)]
505         protected virtual void SetBalanceValue(ref TLink storedValue, sbyte value)
506         {
507             unchecked
508             {
509                 var packagedValue = _int32ToAddressConverter.Convert((byte)value >> 5 & 4 |
                        ↪  value & 3);
510                 var modified = Bit<TLink>.PartialWrite(storedValue, packagedValue, 0, 3);
511                 storedValue = modified;
```

```
512             }
513         }
514 
515         /// <summary>
516         /// <para>
517         /// The zero.
518         /// </para>
519         /// <para></para>
520         /// </summary>
521         public TLink this[TLink index]
522         {
523             [MethodImpl(MethodImplOptions.AggressiveInlining)]
524             get
525             {
526                 var root = GetTreeRoot();
527                 if (GreaterOrEqualThan(index, GetSize(root)))
528                 {
529                     return Zero;
530                 }
531                 while (!EqualToZero(root))
532                 {
533                     var left = GetLeftOrDefault(root);
534                     var leftSize = GetSizeOrZero(left);
535                     if (LessThan(index, leftSize))
536                     {
537                         root = left;
538                         continue;
539                     }
540                     if (AreEqual(index, leftSize))
541                     {
542                         return root;
543                     }
544                     root = GetRightOrDefault(root);
545                     index = Subtract(index, Increment(leftSize));
546                 }
547                 return Zero; // TODO: Impossible situation exception (only if tree structure
                     ↪  broken)
548             }
549         }
550 
551         /// <summary>
552         /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
                 ↪  (концом).
553         /// </summary>
554         /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
555         /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
556         /// <returns>Индекс искомой связи.</returns>
557         [MethodImpl(MethodImplOptions.AggressiveInlining)]
558         public TLink Search(TLink source, TLink target)
559         {
560             var root = GetTreeRoot();
561             while (!EqualToZero(root))
562             {
563                 ref var rootLink = ref GetLinkReference(root);
564                 var rootSource = rootLink.Source;
565                 var rootTarget = rootLink.Target;
566                 if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
                     ↪  node.Key < root.Key
567                 {
568                     root = GetLeftOrDefault(root);
569                 }
570                 else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
                     ↪  node.Key > root.Key
571                 {
572                     root = GetRightOrDefault(root);
573                 }
574                 else // node.Key == root.Key
575                 {
576                     return root;
577                 }
578             }
579             return Zero;
580         }
581 
582         // TODO: Return indices range instead of references count
583         /// <summary>
584         /// <para>
585         /// 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>
```

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

```
        /// <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>
        /// Determines whether this instance first is to the right 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 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>
```

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

```
245         protected override bool FirstIsToTheLeftOfSecond(TLink first, TLink second)
246         {
247             ref var firstLink = ref GetLinkReference(first);
248             ref var secondLink = ref GetLinkReference(second);
249             return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
                ↪  secondLink.Source, secondLink.Target);
250         }
251 
252         /// <summary>
253         /// <para>
254         /// Determines whether this instance first is to the right of second.
255         /// </para>
256         /// <para></para>
257         /// </summary>
258         /// <param name="first">
259         /// <para>The first.</para>
260         /// <para></para>
261         /// </param>
262         /// <param name="second">
263         /// <para>The second.</para>
264         /// <para></para>
265         /// </param>
266         /// <returns>
267         /// <para>The bool</para>
268         /// <para></para>
269         /// </returns>
270         [MethodImpl(MethodImplOptions.AggressiveInlining)]
271         protected override bool FirstIsToTheRightOfSecond(TLink first, TLink second)
272         {
273             ref var firstLink = ref GetLinkReference(first);
274             ref var secondLink = ref GetLinkReference(second);
275             return FirstIsToTheRightOfSecond(firstLink.Source, firstLink.Target,
                ↪  secondLink.Source, secondLink.Target);
276         }
277 
278         /// <summary>
279         /// <para>
280         /// The zero.
281         /// </para>
282         /// <para></para>
283         /// </summary>
284         public TLink this[TLink index]
285         {
286             [MethodImpl(MethodImplOptions.AggressiveInlining)]
287             get
288             {
289                 var root = GetTreeRoot();
290                 if (GreaterOrEqualThan(index, GetSize(root)))
291                 {
292                     return Zero;
293                 }
294                 while (!EqualToZero(root))
295                 {
296                     var left = GetLeftOrDefault(root);
297                     var leftSize = GetSizeOrZero(left);
298                     if (LessThan(index, leftSize))
299                     {
300                         root = left;
301                         continue;
302                     }
303                     if (AreEqual(index, leftSize))
304                     {
305                         return root;
306                     }
307                     root = GetRightOrDefault(root);
308                     index = Subtract(index, Increment(leftSize));
309                 }
310                 return Zero; // TODO: Impossible situation exception (only if tree structure
                    ↪  broken)
311             }
312         }
313 
314         /// <summary>
315         /// Выполняет поиск и возвращает индекс связи с указанными Source (началом) и Target
            ↪  (концом).
316         /// </summary>
317         /// <param name="source">Индекс связи, которая является началом на искомой связи.</param>
318         /// <param name="target">Индекс связи, которая является концом на искомой связи.</param>
```

```
319         /// <returns>Индекс искомой связи.</returns>
320         [MethodImpl(MethodImplOptions.AggressiveInlining)]
321         public TLink Search(TLink source, TLink target)
322         {
323             var root = GetTreeRoot();
324             while (!EqualToZero(root))
325             {
326                 ref var rootLink = ref GetLinkReference(root);
327                 var rootSource = rootLink.Source;
328                 var rootTarget = rootLink.Target;
329                 if (FirstIsToTheLeftOfSecond(source, target, rootSource, rootTarget)) //
                    ↪ node.Key < root.Key
330                 {
331                     root = GetLeftOrDefault(root);
332                 }
333                 else if (FirstIsToTheRightOfSecond(source, target, rootSource, rootTarget)) //
                    ↪ node.Key > root.Key
334                 {
335                     root = GetRightOrDefault(root);
336                 }
337                 else // node.Key == root.Key
338                 {
339                     return root;
340                 }
341             }
342             return Zero;
343         }
344 
345         // TODO: Return indices range instead of references count
346         /// <summary>
347         /// <para>
348         /// Counts the usages using the specified link.
349         /// </para>
350         /// <para></para>
351         /// </summary>
352         /// <param name="link">
353         /// <para>The link.</para>
354         /// <para></para>
355         /// </param>
356         /// <returns>
357         /// <para>The link</para>
358         /// <para></para>
359         /// </returns>
360         [MethodImpl(MethodImplOptions.AggressiveInlining)]
361         public TLink CountUsages(TLink link)
362         {
363             var root = GetTreeRoot();
364             var total = GetSize(root);
365             var totalRightIgnore = Zero;
366             while (!EqualToZero(root))
367             {
368                 var @base = GetBasePartValue(root);
369                 if (LessOrEqualThan(@base, link))
370                 {
371                     root = GetRightOrDefault(root);
372                 }
373                 else
374                 {
375                     totalRightIgnore = Add(totalRightIgnore, Increment(GetRightSize(root)));
376                     root = GetLeftOrDefault(root);
377                 }
378             }
379             root = GetTreeRoot();
380             var totalLeftIgnore = Zero;
381             while (!EqualToZero(root))
382             {
383                 var @base = GetBasePartValue(root);
384                 if (GreaterOrEqualThan(@base, link))
385                 {
386                     root = GetLeftOrDefault(root);
387                 }
388                 else
389                 {
390                     totalLeftIgnore = Add(totalLeftIgnore, Increment(GetLeftSize(root)));
391                     root = GetRightOrDefault(root);
392                 }
393             }
394             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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        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>
        /// <para></para>
        /// </summary>
        protected readonly byte* Header;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksSizeBalancedTreeMethodsBase"/> 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 LinksSizeBalancedTreeMethodsBase(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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

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

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

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

```
                {
                    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)]
```

```
416         public TLink EachUsage(TLink @base, Func<IList<TLink>, TLink> handler) =>
            ↪  EachUsageCore(@base, GetTreeRoot(), handler);
417
418         // TODO: 1. Move target, handler to separate object. 2. Use stack or walker 3. Use
            ↪  low-level MSIL stack.
419         /// <summary>
420         /// <para>
421         /// Eaches the usage core using the specified base.
422         /// </para>
423         /// <para></para>
424         /// </summary>
425         /// <param name="@base">
426         /// <para>The base.</para>
427         /// <para></para>
428         /// </param>
429         /// <param name="link">
430         /// <para>The link.</para>
431         /// <para></para>
432         /// </param>
433         /// <param name="handler">
434         /// <para>The handler.</para>
435         /// <para></para>
436         /// </param>
437         /// <returns>
438         /// <para>The continue.</para>
439         /// <para></para>
440         /// </returns>
441         [MethodImpl(MethodImplOptions.AggressiveInlining)]
442         private TLink EachUsageCore(TLink @base, TLink link, Func<IList<TLink>, TLink> handler)
443         {
444             var @continue = Continue;
445             if (EqualToZero(link))
446             {
447                 return @continue;
448             }
449             var linkBasePart = GetBasePartValue(link);
450             var @break = Break;
451             if (GreaterThan(linkBasePart, @base))
452             {
453                 if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
454                 {
455                     return @break;
456                 }
457             }
458             else if (LessThan(linkBasePart, @base))
459             {
460                 if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
461                 {
462                     return @break;
463                 }
464             }
465             else //if (linkBasePart == @base)
466             {
467                 if (AreEqual(handler(GetLinkValues(link)), @break))
468                 {
469                     return @break;
470                 }
471                 if (AreEqual(EachUsageCore(@base, GetLeftOrDefault(link), handler), @break))
472                 {
473                     return @break;
474                 }
475                 if (AreEqual(EachUsageCore(@base, GetRightOrDefault(link), handler), @break))
476                 {
477                     return @break;
478                 }
479             }
480             return @continue;
481         }
482
483         /// <summary>
484         /// <para>
485         /// Prints the node value using the specified node.
486         /// </para>
487         /// <para></para>
488         /// </summary>
489         /// <param name="node">
490         /// <para>The node.</para>
491         /// <para></para>
```

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

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

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

```
        /// </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>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override TLink GetSize(TLink node) =>
        ↪  GetSizeValue(GetLinkReference(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) => SetSizeValue(ref
        ↪  GetLinkReference(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>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GetLeftIsChild(TLink node) =>
        ↪  GetLeftIsChildValue(GetLinkReference(node).SizeAsSource);

        /// <summary>
        /// <para>
        /// Sets the left is child 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 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>
```

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

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

```
        /// </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;
        }
    }
}
```

1.82 ./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

namespace Platform.Data.Doublets.Memory.United.Generic
{
    /// <summary>
    /// <para>
    /// Represents the links sources size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksSizeBalancedTreeMethodsBase{TLink}"/>
    public unsafe class LinksSourcesSizeBalancedTreeMethods<TLink> :
    ↪ LinksSizeBalancedTreeMethodsBase<TLink>
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksSourcesSizeBalancedTreeMethods"/> 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 LinksSourcesSizeBalancedTreeMethods(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;
        }
    }
}
```

1.83 ./csharp/Platform.Data.Doublets/Memory/United/Generic/LinksTargetsAvlBalancedTreeMethods.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 targets avl balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="LinksAvlBalancedTreeMethodsBase{TLink}"/>
14     public unsafe class LinksTargetsAvlBalancedTreeMethods<TLink> :
       ↪ LinksAvlBalancedTreeMethodsBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="LinksTargetsAvlBalancedTreeMethods"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="constants">
23         /// <para>A constants.</para>
24         /// <para></para>
25         /// </param>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>
32         /// <para></para>
33         /// </param>
34         [MethodImpl(MethodImplOptions.AggressiveInlining)]
35         public LinksTargetsAvlBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
           ↪ byte* header) : base(constants, links, header) { }
36
37         /// <summary>
38         /// <para>
39         /// Gets the left reference using the specified node.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         /// <param name="node">
44         /// <para>The node.</para>
45         /// <para></para>
46         /// </param>
47         /// <returns>
48         /// <para>The ref link</para>
49         /// <para></para>
50         /// </returns>
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         protected override ref TLink GetLeftReference(TLink node) => ref
           ↪ GetLinkReference(node).LeftAsTarget;
53
54         /// <summary>
55         /// <para>
56         /// Gets the right reference using the specified node.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         /// <param name="node">
61         /// <para>The node.</para>
62         /// <para></para>
63         /// </param>
64         /// <returns>
65         /// <para>The ref link</para>
66         /// <para></para>
67         /// </returns>
68         [MethodImpl(MethodImplOptions.AggressiveInlining)]
69         protected override ref TLink GetRightReference(TLink node) => ref
           ↪ GetLinkReference(node).RightAsTarget;
70
71         /// <summary>
72         /// <para>
73         /// Gets the left using the specified node.
74         /// </para>
75         /// <para></para>
76         /// </summary>
77         /// <param name="node">
78         /// <para>The node.</para>
```

```
        /// <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)]
```

```
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>
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).SizeAsTarget);
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).SizeAsTarget, value);
274
275         /// <summary>
276         /// <para>
277         /// Gets the tree root.
278         /// </para>
279         /// <para></para>
280         /// </summary>
281         /// <returns>
282         /// <para>The link</para>
283         /// <para></para>
284         /// </returns>
285         [MethodImpl(MethodImplOptions.AggressiveInlining)]
286         protected override TLink GetTreeRoot() => GetHeaderReference().RootAsTarget;
287
288         /// <summary>
289         /// <para>
290         /// Gets the base part value using the specified link.
291         /// </para>
292         /// <para></para>
293         /// </summary>
294         /// <param name="link">
295         /// <para>The link.</para>
296         /// <para></para>
297         /// </param>
298         /// <returns>
299         /// <para>The link</para>
300         /// <para></para>
```

```
        /// </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)]
```

```
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>
9      /// Represents the links targets recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{TLink}"/>
14     public unsafe class LinksTargetsRecursionlessSizeBalancedTreeMethods<TLink> :
       ↪  LinksRecursionlessSizeBalancedTreeMethodsBase<TLink>
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="LinksTargetsRecursionlessSizeBalancedTreeMethods"/>
           ↪  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 LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<TLink> constants,
           ↪  byte* links, byte* header) : base(constants, links, header) { }
36
37         /// <summary>
38         /// <para>
39         /// Gets the left reference using the specified node.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         /// <param name="node">
44         /// <para>The node.</para>
45         /// <para></para>
46         /// </param>
47         /// <returns>
48         /// <para>The ref link</para>
49         /// <para></para>
50         /// </returns>
51         [MethodImpl(MethodImplOptions.AggressiveInlining)]
52         protected override ref TLink GetLeftReference(TLink node) => ref
           ↪  GetLinkReference(node).LeftAsTarget;
53
54         /// <summary>
55         /// <para>
56         /// Gets the right reference using the specified node.
57         /// </para>
58         /// <para></para>
59         /// </summary>
60         /// <param name="node">
61         /// <para>The node.</para>
62         /// <para></para>
63         /// </param>
```

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

```
        /// <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;
        }
    }
}
```

1.85 ./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>
```

```
8       /// <para>
9       /// Represents the links targets size balanced tree methods.
10      /// </para>
11      /// <para></para>
12      /// </summary>
13      /// <seealso cref="LinksSizeBalancedTreeMethodsBase{TLink}"/>
14      public unsafe class LinksTargetsSizeBalancedTreeMethods<TLink> :
        ↪  LinksSizeBalancedTreeMethodsBase<TLink>
15      {
16          /// <summary>
17          /// <para>
18          /// Initializes a new <see cref="LinksTargetsSizeBalancedTreeMethods"/> instance.
19          /// </para>
20          /// <para></para>
21          /// </summary>
22          /// <param name="constants">
23          /// <para>A constants.</para>
24          /// <para></para>
25          /// </param>
26          /// <param name="links">
27          /// <para>A links.</para>
28          /// <para></para>
29          /// </param>
30          /// <param name="header">
31          /// <para>A header.</para>
32          /// <para></para>
33          /// </param>
34          [MethodImpl(MethodImplOptions.AggressiveInlining)]
35          public LinksTargetsSizeBalancedTreeMethods(LinksConstants<TLink> constants, byte* links,
            ↪  byte* header) : base(constants, links, header) { }
36
37          /// <summary>
38          /// <para>
39          /// Gets the left reference using the specified node.
40          /// </para>
41          /// <para></para>
42          /// </summary>
43          /// <param name="node">
44          /// <para>The node.</para>
45          /// <para></para>
46          /// </param>
47          /// <returns>
48          /// <para>The ref link</para>
49          /// <para></para>
50          /// </returns>
51          [MethodImpl(MethodImplOptions.AggressiveInlining)]
52          protected override ref TLink GetLeftReference(TLink node) => ref
            ↪  GetLinkReference(node).LeftAsTarget;
53
54          /// <summary>
55          /// <para>
56          /// Gets the right reference using the specified node.
57          /// </para>
58          /// <para></para>
59          /// </summary>
60          /// <param name="node">
61          /// <para>The node.</para>
62          /// <para></para>
63          /// </param>
64          /// <returns>
65          /// <para>The ref link</para>
66          /// <para></para>
67          /// </returns>
68          [MethodImpl(MethodImplOptions.AggressiveInlining)]
69          protected override ref TLink GetRightReference(TLink node) => ref
            ↪  GetLinkReference(node).RightAsTarget;
70
71          /// <summary>
72          /// <para>
73          /// Gets the left using the specified node.
74          /// </para>
75          /// <para></para>
76          /// </summary>
77          /// <param name="node">
78          /// <para>The node.</para>
79          /// <para></para>
80          /// </param>
81          /// <returns>
```

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

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

1.86 ./csharp/Platform.Data.Doublets/Memory/United/Generic/UnitedMemoryLinks.cs

```
1   using System;
2   using System.Runtime.CompilerServices;
3   using Platform.Singletons;
4   using Platform.Memory;
5   using static System.Runtime.CompilerServices.Unsafe;
6
7   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9   namespace Platform.Data.Doublets.Memory.United.Generic
10  {
11      /// <summary>
12      /// <para>
13      /// Represents the united memory links.
14      /// </para>
15      /// <para></para>
16      /// </summary>
17      /// <seealso cref="UnitedMemoryLinksBase{TLink}"/>
18      public unsafe class UnitedMemoryLinks<TLink> : UnitedMemoryLinksBase<TLink>
19      {
20          /// <summary>
21          /// <para>
22          /// The create source tree methods.
23          /// </para>
24          /// <para></para>
25          /// </summary>
26          private readonly Func<ILinksTreeMethods<TLink>> _createSourceTreeMethods;
27          /// <summary>
```

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

```
99
100         /// <summary>
101         /// <para>
102         /// Initializes a new <see cref="UnitedMemoryLinks"/> instance.
103         /// </para>
104         /// <para></para>
105         /// </summary>
106         /// <param name="memory">
107         /// <para>A memory.</para>
108         /// <para></para>
109         /// </param>
110         /// <param name="memoryReservationStep">
111         /// <para>A memory reservation step.</para>
112         /// <para></para>
113         /// </param>
114         /// <param name="constants">
115         /// <para>A constants.</para>
116         /// <para></para>
117         /// </param>
118         /// <param name="indexTreeType">
119         /// <para>A index tree type.</para>
120         /// <para></para>
121         /// </param>
122         [MethodImpl(MethodImplOptions.AggressiveInlining)]
123         public UnitedMemoryLinks(IResizableDirectMemory memory, long memoryReservationStep,
            ↪ LinksConstants<TLink> constants, IndexTreeType indexTreeType) : base(memory,
            ↪ memoryReservationStep, constants)
124         {
125             if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
126             {
127                 _createSourceTreeMethods = () => new
                    ↪ LinksSourcesAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
128                 _createTargetTreeMethods = () => new
                    ↪ LinksTargetsAvlBalancedTreeMethods<TLink>(Constants, _links, _header);
129             }
130             else
131             {
132                 _createSourceTreeMethods = () => new
                    ↪ LinksSourcesSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
133                 _createTargetTreeMethods = () => new
                    ↪ LinksTargetsSizeBalancedTreeMethods<TLink>(Constants, _links, _header);
134             }
135             Init(memory, memoryReservationStep);
136         }
137
138         /// <summary>
139         /// <para>
140         /// Sets the pointers using the specified memory.
141         /// </para>
142         /// <para></para>
143         /// </summary>
144         /// <param name="memory">
145         /// <para>The memory.</para>
146         /// <para></para>
147         /// </param>
148         [MethodImpl(MethodImplOptions.AggressiveInlining)]
149         protected override void SetPointers(IResizableDirectMemory memory)
150         {
151             _links = (byte*)memory.Pointer;
152             _header = _links;
153             SourcesTreeMethods = _createSourceTreeMethods();
154             TargetsTreeMethods = _createTargetTreeMethods();
155             UnusedLinksListMethods = new UnusedLinksListMethods<TLink>(_links, _header);
156         }
157
158         /// <summary>
159         /// <para>
160         /// Resets the pointers.
161         /// </para>
162         /// <para></para>
163         /// </summary>
164         [MethodImpl(MethodImplOptions.AggressiveInlining)]
165         protected override void ResetPointers()
166         {
167             base.ResetPointers();
168             _links = null;
169             _header = null;
170         }
```

```

        /// <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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly Comparer<TLink> _comparer = Comparer<TLink>.Default;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
```

```
43          /// <para></para>
44          /// </summary>
45          private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
            ↪  UncheckedConverter<TLink, long>.Default;
46          /// <summary>
47          /// <para>
48          /// The default.
49          /// </para>
50          /// <para></para>
51          /// </summary>
52          private static readonly UncheckedConverter<long, TLink> _int64ToAddressConverter =
            ↪  UncheckedConverter<long, TLink>.Default;
53
54          /// <summary>
55          /// <para>
56          /// The zero.
57          /// </para>
58          /// <para></para>
59          /// </summary>
60          private static readonly TLink _zero = default;
61          /// <summary>
62          /// <para>
63          /// The zero.
64          /// </para>
65          /// <para></para>
66          /// </summary>
67          private static readonly TLink _one = Arithmetic.Increment(_zero);
68
69          /// <summary>Возвращает размер одной связи в байтах.</summary>
70          /// <remarks>
71          /// Используется только во вне класса, не рекомедуется использовать внутри.
72          /// Так как во вне не обязательно будет доступен unsafe C#.
73          /// </remarks>
74          public static readonly long LinkSizeInBytes = RawLink<TLink>.SizeInBytes;
75
76          /// <summary>
77          /// <para>
78          /// The size in bytes.
79          /// </para>
80          /// <para></para>
81          /// </summary>
82          public static readonly long LinkHeaderSizeInBytes = LinksHeader<TLink>.SizeInBytes;
83
84          /// <summary>
85          /// <para>
86          /// The link size in bytes.
87          /// </para>
88          /// <para></para>
89          /// </summary>
90          public static readonly long DefaultLinksSizeStep = LinkSizeInBytes * 1024 * 1024;
91
92          /// <summary>
93          /// <para>
94          /// The memory.
95          /// </para>
96          /// <para></para>
97          /// </summary>
98          protected readonly IResizableDirectMemory _memory;
99          /// <summary>
100         /// <para>
101         /// The memory reservation step.
102         /// </para>
103         /// <para></para>
104         /// </summary>
105         protected readonly long _memoryReservationStep;
106
107         /// <summary>
108         /// <para>
109         /// The targets tree methods.
110         /// </para>
111         /// <para></para>
112         /// </summary>
113         protected ILinksTreeMethods<TLink> TargetsTreeMethods;
114         /// <summary>
115         /// <para>
116         /// The sources tree methods.
117         /// </para>
118         /// <para></para>
119         /// </summary>
```

```
120         protected ILinksTreeMethods<TLink> SourcesTreeMethods;
121         // TODO: Возможно чтобы гарантированно проверять на то, является ли связь удалённой,
            ↪  нужно использовать не список а дерево, так как так можно быстрее проверить на
            ↪  наличие связи внутри
122         /// <summary>
123         /// <para>
124         /// The unused links list methods.
125         /// </para>
126         /// <para></para>
127         /// </summary>
128         protected ILinksListMethods<TLink> UnusedLinksListMethods;
129
130         /// <summary>
131         /// Возвращает общее число связей находящихся в хранилище.
132         /// </summary>
133         protected virtual TLink Total
134         {
135             [MethodImpl(MethodImplOptions.AggressiveInlining)]
136             get
137             {
138                 ref var header = ref GetHeaderReference();
139                 return Subtract(header.AllocatedLinks, header.FreeLinks);
140             }
141         }
142
143         /// <summary>
144         /// <para>
145         /// Gets the constants value.
146         /// </para>
147         /// <para></para>
148         /// </summary>
149         public virtual LinksConstants<TLink> Constants
150         {
151             [MethodImpl(MethodImplOptions.AggressiveInlining)]
152             get;
153         }
154
155         /// <summary>
156         /// <para>
157         /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
158         /// </para>
159         /// <para></para>
160         /// </summary>
161         /// <param name="memory">
162         /// <para>A memory.</para>
163         /// <para></para>
164         /// </param>
165         /// <param name="memoryReservationStep">
166         /// <para>A memory reservation step.</para>
167         /// <para></para>
168         /// </param>
169         /// <param name="constants">
170         /// <para>A constants.</para>
171         /// <para></para>
172         /// </param>
173         [MethodImpl(MethodImplOptions.AggressiveInlining)]
174         protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
            ↪  memoryReservationStep, LinksConstants<TLink> constants)
175         {
176             _memory = memory;
177             _memoryReservationStep = memoryReservationStep;
178             Constants = constants;
179         }
180
181         /// <summary>
182         /// <para>
183         /// Initializes a new <see cref="UnitedMemoryLinksBase"/> instance.
184         /// </para>
185         /// <para></para>
186         /// </summary>
187         /// <param name="memory">
188         /// <para>A memory.</para>
189         /// <para></para>
190         /// </param>
191         /// <param name="memoryReservationStep">
192         /// <para>A memory reservation step.</para>
193         /// <para></para>
194         /// </param>
195         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
196         protected UnitedMemoryLinksBase(IResizableDirectMemory memory, long
            ↪ memoryReservationStep) : this(memory, memoryReservationStep,
            ↪ Default<LinksConstants<TLink>>.Instance) { }
197
198         /// <summary>
199         /// <para>
200         /// Inits the memory.
201         /// </para>
202         /// <para></para>
203         /// </summary>
204         /// <param name="memory">
205         /// <para>The memory.</para>
206         /// <para></para>
207         /// </param>
208         /// <param name="memoryReservationStep">
209         /// <para>The memory reservation step.</para>
210         /// <para></para>
211         /// </param>
212         [MethodImpl(MethodImplOptions.AggressiveInlining)]
213         protected virtual void Init(IResizableDirectMemory memory, long memoryReservationStep)
214         {
215             if (memory.ReservedCapacity < memoryReservationStep)
216             {
217                 memory.ReservedCapacity = memoryReservationStep;
218             }
219             SetPointers(memory);
220             ref var header = ref GetHeaderReference();
221             // Гарантия корректности _memory.UsedCapacity относительно _header->AllocatedLinks
222             memory.UsedCapacity = (ConvertToInt64(header.AllocatedLinks) * LinkSizeInBytes) +
                ↪ LinkHeaderSizeInBytes;
223             // Гарантия корректности _header->ReservedLinks относительно _memory.ReservedCapacity
224             header.ReservedLinks = ConvertToAddress((memory.ReservedCapacity -
                ↪ LinkHeaderSizeInBytes) / LinkSizeInBytes);
225         }
226
227         /// <summary>
228         /// <para>
229         /// Counts the restrictions.
230         /// </para>
231         /// <para></para>
232         /// </summary>
233         /// <param name="restrictions">
234         /// <para>The restrictions.</para>
235         /// <para></para>
236         /// </param>
237         /// <exception cref="NotSupportedException">
238         /// <para>Другие размеры и способы ограничений не поддерживаются.</para>
239         /// <para></para>
240         /// </exception>
241         /// <returns>
242         /// <para>The link</para>
243         /// <para></para>
244         /// </returns>
245         [MethodImpl(MethodImplOptions.AggressiveInlining)]
246         public virtual TLink Count(IList<TLink> restrictions)
247         {
248             // Если нет ограничений, тогда возвращаем общее число связей находящихся в хранилище.
249             if (restrictions.Count == 0)
250             {
251                 return Total;
252             }
253             var constants = Constants;
254             var any = constants.Any;
255             var index = restrictions[constants.IndexPart];
256             if (restrictions.Count == 1)
257             {
258                 if (AreEqual(index, any))
259                 {
260                     return Total;
261                 }
262                 return Exists(index) ? GetOne() : GetZero();
263             }
264             if (restrictions.Count == 2)
265             {
266                 var value = restrictions[1];
267                 if (AreEqual(index, any))
268                 {
269                     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) ||
```

```
497                         AreEqual(storedLinkValue.Target, value))
498                     {
499                         return handler(GetLinkStruct(index));
500                     }
501                     return @continue;
502                 }
503             }
504             throw new NotSupportedException("Другие размеры и способы ограничений не
                ↪ поддерживаются.");
505         }
506 
507         /// <remarks>
508         /// TODO: Возможно можно перемещать значения, если указан индекс, но значение существует
                ↪ в другом месте (но не в менеджере памяти, а в логике Links)
509         /// </remarks>
510         [MethodImpl(MethodImplOptions.AggressiveInlining)]
511         public virtual TLink Update(IList<TLink> restrictions, IList<TLink> substitution)
512         {
513             var constants = Constants;
514             var @null = constants.Null;
515             var linkIndex = restrictions[constants.IndexPart];
516             ref var link = ref GetLinkReference(linkIndex);
517             ref var header = ref GetHeaderReference();
518             ref var firstAsSource = ref header.RootAsSource;
519             ref var firstAsTarget = ref header.RootAsTarget;
520             // Будет корректно работать только в том случае, если пространство выделенной связи
                ↪ предварительно заполнено нулями
521             if (!AreEqual(link.Source, @null))
522             {
523                 SourcesTreeMethods.Detach(ref firstAsSource, linkIndex);
524             }
525             if (!AreEqual(link.Target, @null))
526             {
527                 TargetsTreeMethods.Detach(ref firstAsTarget, linkIndex);
528             }
529             link.Source = substitution[constants.SourcePart];
530             link.Target = substitution[constants.TargetPart];
531             if (!AreEqual(link.Source, @null))
532             {
533                 SourcesTreeMethods.Attach(ref firstAsSource, linkIndex);
534             }
535             if (!AreEqual(link.Target, @null))
536             {
537                 TargetsTreeMethods.Attach(ref firstAsTarget, linkIndex);
538             }
539             return linkIndex;
540         }
541 
542         /// <remarks>
543         /// TODO: Возможно нужно будет заполнение нулями, если внешнее API ими не заполняет
                ↪ пространство
544         /// </remarks>
545         [MethodImpl(MethodImplOptions.AggressiveInlining)]
546         public virtual TLink Create(IList<TLink> restrictions)
547         {
548             ref var header = ref GetHeaderReference();
549             var freeLink = header.FirstFreeLink;
550             if (!AreEqual(freeLink, Constants.Null))
551             {
552                 UnusedLinksListMethods.Detach(freeLink);
553             }
554             else
555             {
556                 var maximumPossibleInnerReference = Constants.InternalReferencesRange.Maximum;
557                 if (GreaterThan(header.AllocatedLinks, maximumPossibleInnerReference))
558                 {
559                     throw new LinksLimitReachedException<TLink>(maximumPossibleInnerReference);
560                 }
561                 if (GreaterOrEqualThan(header.AllocatedLinks, Decrement(header.ReservedLinks)))
562                 {
563                     _memory.ReservedCapacity += _memoryReservationStep;
564                     SetPointers(_memory);
565                     header = ref GetHeaderReference();
566                     header.ReservedLinks = ConvertToAddress(_memory.ReservedCapacity /
                        ↪ LinkSizeInBytes);
567                 }
568                 freeLink = header.AllocatedLinks = Increment(header.AllocatedLinks);
569                 _memory.UsedCapacity += LinkSizeInBytes;
570             }
```

```
571             return freeLink;
572         }
573 
574         /// <summary>
575         /// <para>
576         /// Deletes the restrictions.
577         /// </para>
578         /// <para></para>
579         /// </summary>
580         /// <param name="restrictions">
581         /// <para>The restrictions.</para>
582         /// <para></para>
583         /// </param>
584         [MethodImpl(MethodImplOptions.AggressiveInlining)]
585         public virtual void Delete(IList<TLink> restrictions)
586         {
587             ref var header = ref GetHeaderReference();
588             var link = restrictions[Constants.IndexPart];
589             if (LessThan(link, header.AllocatedLinks))
590             {
591                 UnusedLinksListMethods.AttachAsFirst(link);
592             }
593             else if (AreEqual(link, header.AllocatedLinks))
594             {
595                 header.AllocatedLinks = Decrement(header.AllocatedLinks);
596                 _memory.UsedCapacity -= LinkSizeInBytes;
597                 // Убираем все связи, находящиеся в списке свободных в конце файла, до тех пор,
                    ↪  пока не дойдём до первой существующей связи
598                 // Позволяет оптимизировать количество выделенных связей (AllocatedLinks)
599                 while (GreaterThan(header.AllocatedLinks, GetZero()) &&
                    ↪  IsUnusedLink(header.AllocatedLinks))
600                 {
601                     UnusedLinksListMethods.Detach(header.AllocatedLinks);
602                     header.AllocatedLinks = Decrement(header.AllocatedLinks);
603                     _memory.UsedCapacity -= LinkSizeInBytes;
604                 }
605             }
606         }
607 
608         /// <summary>
609         /// <para>
610         /// Gets the link struct using the specified link index.
611         /// </para>
612         /// <para></para>
613         /// </summary>
614         /// <param name="linkIndex">
615         /// <para>The link index.</para>
616         /// <para></para>
617         /// </param>
618         /// <returns>
619         /// <para>A list of t link</para>
620         /// <para></para>
621         /// </returns>
622         [MethodImpl(MethodImplOptions.AggressiveInlining)]
623         public IList<TLink> GetLinkStruct(TLink linkIndex)
624         {
625             ref var link = ref GetLinkReference(linkIndex);
626             return new Link<TLink>(linkIndex, link.Source, link.Target);
627         }
628 
629         /// <remarks>
630         /// TODO: Возможно это должно быть событием, вызываемым из IMemory, в том случае, если
                ↪  адрес реально поменялся
631         ///
632         /// Указатель this.links может быть в том же месте,
633         /// так как 0-я связь не используется и имеет такой же размер как Header,
634         /// поэтому header размещается в том же месте, что и 0-я связь
635         /// </remarks>
636         [MethodImpl(MethodImplOptions.AggressiveInlining)]
637         protected abstract void SetPointers(IResizableDirectMemory memory);
638 
639         /// <summary>
640         /// <para>
641         /// Resets the pointers.
642         /// </para>
643         /// <para></para>
644         /// </summary>
645         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
646         protected virtual void ResetPointers()
647         {
648             SourcesTreeMethods = null;
649             TargetsTreeMethods = null;
650             UnusedLinksListMethods = null;
651         }
652 
653         /// <summary>
654         /// <para>
655         /// Gets the header reference.
656         /// </para>
657         /// <para></para>
658         /// </summary>
659         /// <returns>
660         /// <para>A ref links header of t link</para>
661         /// <para></para>
662         /// </returns>
663         [MethodImpl(MethodImplOptions.AggressiveInlining)]
664         protected abstract ref LinksHeader<TLink> GetHeaderReference();
665 
666         /// <summary>
667         /// <para>
668         /// Gets the link reference using the specified link index.
669         /// </para>
670         /// <para></para>
671         /// </summary>
672         /// <param name="linkIndex">
673         /// <para>The link index.</para>
674         /// <para></para>
675         /// </param>
676         /// <returns>
677         /// <para>A ref raw link of t link</para>
678         /// <para></para>
679         /// </returns>
680         [MethodImpl(MethodImplOptions.AggressiveInlining)]
681         protected abstract ref RawLink<TLink> GetLinkReference(TLink linkIndex);
682 
683         /// <summary>
684         /// <para>
685         /// Determines whether this instance exists.
686         /// </para>
687         /// <para></para>
688         /// </summary>
689         /// <param name="link">
690         /// <para>The link.</para>
691         /// <para></para>
692         /// </param>
693         /// <returns>
694         /// <para>The bool</para>
695         /// <para></para>
696         /// </returns>
697         [MethodImpl(MethodImplOptions.AggressiveInlining)]
698         protected virtual bool Exists(TLink link)
699             => GreaterOrEqualThan(link, Constants.InternalReferencesRange.Minimum)
700             && LessOrEqualThan(link, GetHeaderReference().AllocatedLinks)
701             && !IsUnusedLink(link);
702 
703         /// <summary>
704         /// <para>
705         /// Determines whether this instance is unused link.
706         /// </para>
707         /// <para></para>
708         /// </summary>
709         /// <param name="linkIndex">
710         /// <para>The link index.</para>
711         /// <para></para>
712         /// </param>
713         /// <returns>
714         /// <para>The bool</para>
715         /// <para></para>
716         /// </returns>
717         [MethodImpl(MethodImplOptions.AggressiveInlining)]
718         protected virtual bool IsUnusedLink(TLink linkIndex)
719         {
720             if (!AreEqual(GetHeaderReference().FirstFreeLink, linkIndex)) // May be this check
                ↪  is not needed
721             {
722                 ref var link = ref GetLinkReference(linkIndex);
```

```
723                 return AreEqual(link.SizeAsSource, default) && !AreEqual(link.Source, default);
724             }
725             else
726             {
727                 return true;
728             }
729         }
730 
731         /// <summary>
732         /// <para>
733         /// Gets the one.
734         /// </para>
735         /// <para></para>
736         /// </summary>
737         /// <returns>
738         /// <para>The link</para>
739         /// <para></para>
740         /// </returns>
741         [MethodImpl(MethodImplOptions.AggressiveInlining)]
742         protected virtual TLink GetOne() => _one;
743 
744         /// <summary>
745         /// <para>
746         /// Gets the zero.
747         /// </para>
748         /// <para></para>
749         /// </summary>
750         /// <returns>
751         /// <para>The link</para>
752         /// <para></para>
753         /// </returns>
754         [MethodImpl(MethodImplOptions.AggressiveInlining)]
755         protected virtual TLink GetZero() => default;
756 
757         /// <summary>
758         /// <para>
759         /// Determines whether this instance are equal.
760         /// </para>
761         /// <para></para>
762         /// </summary>
763         /// <param name="first">
764         /// <para>The first.</para>
765         /// <para></para>
766         /// </param>
767         /// <param name="second">
768         /// <para>The second.</para>
769         /// <para></para>
770         /// </param>
771         /// <returns>
772         /// <para>The bool</para>
773         /// <para></para>
774         /// </returns>
775         [MethodImpl(MethodImplOptions.AggressiveInlining)]
776         protected virtual bool AreEqual(TLink first, TLink second) =>
            ↪ _equalityComparer.Equals(first, second);
777 
778         /// <summary>
779         /// <para>
780         /// Determines whether this instance less than.
781         /// </para>
782         /// <para></para>
783         /// </summary>
784         /// <param name="first">
785         /// <para>The first.</para>
786         /// <para></para>
787         /// </param>
788         /// <param name="second">
789         /// <para>The second.</para>
790         /// <para></para>
791         /// </param>
792         /// <returns>
793         /// <para>The bool</para>
794         /// <para></para>
795         /// </returns>
796         [MethodImpl(MethodImplOptions.AggressiveInlining)]
797         protected virtual bool LessThan(TLink first, TLink second) => _comparer.Compare(first,
            ↪ second) < 0;
798 
```

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

```
948             /// <returns>
949             /// <para>The link</para>
950             /// <para></para>
951             /// </returns>
952             [MethodImpl(MethodImplOptions.AggressiveInlining)]
953             protected virtual TLink Increment(TLink link) => Arithmetic<TLink>.Increment(link);
954 
955             /// <summary>
956             /// <para>
957             /// Decrements the link.
958             /// </para>
959             /// <para></para>
960             /// </summary>
961             /// <param name="link">
962             /// <para>The link.</para>
963             /// <para></para>
964             /// </param>
965             /// <returns>
966             /// <para>The link</para>
967             /// <para></para>
968             /// </returns>
969             [MethodImpl(MethodImplOptions.AggressiveInlining)]
970             protected virtual TLink Decrement(TLink link) => Arithmetic<TLink>.Decrement(link);
971 
972             #region Disposable
973 
974             /// <summary>
975             /// <para>
976             /// Gets the allow multiple dispose calls value.
977             /// </para>
978             /// <para></para>
979             /// </summary>
980             protected override bool AllowMultipleDisposeCalls
981             {
982                 [MethodImpl(MethodImplOptions.AggressiveInlining)]
983                 get => true;
984             }
985 
986             /// <summary>
987             /// <para>
988             /// Disposes the manual.
989             /// </para>
990             /// <para></para>
991             /// </summary>
992             /// <param name="manual">
993             /// <para>The manual.</para>
994             /// <para></para>
995             /// </param>
996             /// <param name="wasDisposed">
997             /// <para>The was disposed.</para>
998             /// <para></para>
999             /// </param>
1000            [MethodImpl(MethodImplOptions.AggressiveInlining)]
1001            protected override void Dispose(bool manual, bool wasDisposed)
1002            {
1003                if (!wasDisposed)
1004                {
1005                    ResetPointers();
1006                    _memory.DisposeIfPossible();
1007                }
1008            }
1009 
1010            #endregion
1011        }
1012    }
```

1.88 ./csharp/Platform.Data.Doublets/Memory/United/Generic/UnusedLinksListMethods.cs

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

```
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="AbsoluteCircularDoublyLinkedListMethods{TLink}"/>
    /// <seealso cref="ILinksListMethods{TLink}"/>
    public unsafe class UnusedLinksListMethods<TLink> :
    ↪  AbsoluteCircularDoublyLinkedListMethods<TLink>, ILinksListMethods<TLink>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly UncheckedConverter<TLink, long> _addressToInt64Converter =
        ↪  UncheckedConverter<TLink, long>.Default;

        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly byte* _links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        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>
```

```
164             [MethodImpl(MethodImplOptions.AggressiveInlining)]
165             protected override TLink GetSize() => GetHeaderReference().FreeLinks;
166
167             /// <summary>
168             /// <para>
169             /// Sets the first using the specified element.
170             /// </para>
171             /// <para></para>
172             /// </summary>
173             /// <param name="element">
174             /// <para>The element.</para>
175             /// <para></para>
176             /// </param>
177             [MethodImpl(MethodImplOptions.AggressiveInlining)]
178             protected override void SetFirst(TLink element) => GetHeaderReference().FirstFreeLink =
                ↪  element;
179
180             /// <summary>
181             /// <para>
182             /// Sets the last using the specified element.
183             /// </para>
184             /// <para></para>
185             /// </summary>
186             /// <param name="element">
187             /// <para>The element.</para>
188             /// <para></para>
189             /// </param>
190             [MethodImpl(MethodImplOptions.AggressiveInlining)]
191             protected override void SetLast(TLink element) => GetHeaderReference().LastFreeLink =
                ↪  element;
192
193             /// <summary>
194             /// <para>
195             /// Sets the previous using the specified element.
196             /// </para>
197             /// <para></para>
198             /// </summary>
199             /// <param name="element">
200             /// <para>The element.</para>
201             /// <para></para>
202             /// </param>
203             /// <param name="previous">
204             /// <para>The previous.</para>
205             /// <para></para>
206             /// </param>
207             [MethodImpl(MethodImplOptions.AggressiveInlining)]
208             protected override void SetPrevious(TLink element, TLink previous) =>
                ↪  GetLinkReference(element).Source = previous;
209
210             /// <summary>
211             /// <para>
212             /// Sets the next using the specified element.
213             /// </para>
214             /// <para></para>
215             /// </summary>
216             /// <param name="element">
217             /// <para>The element.</para>
218             /// <para></para>
219             /// </param>
220             /// <param name="next">
221             /// <para>The next.</para>
222             /// <para></para>
223             /// </param>
224             [MethodImpl(MethodImplOptions.AggressiveInlining)]
225             protected override void SetNext(TLink element, TLink next) =>
                ↪  GetLinkReference(element).Target = next;
226
227             /// <summary>
228             /// <para>
229             /// Sets the size using the specified size.
230             /// </para>
231             /// <para></para>
232             /// </summary>
233             /// <param name="size">
234             /// <para>The size.</para>
235             /// <para></para>
236             /// </param>
237             [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
238         protected override void SetSize(TLink size) => GetHeaderReference().FreeLinks = size;
239     }
240 }
```

1.89 ./csharp/Platform.Data.Doublets/Memory/United/RawLink.cs

```
1  using Platform.Unsafe;
2  using System;
3  using System.Collections.Generic;
4  using System.Runtime.CompilerServices;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Memory.United
9  {
10     /// <summary>
11     /// <para>
12     /// The raw link.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public struct RawLink<TLink> : IEquatable<RawLink<TLink>>
17     {
18         /// <summary>
19         /// <para>
20         /// The default.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         private static readonly EqualityComparer<TLink> _equalityComparer =
           ↪  EqualityComparer<TLink>.Default;
25
26         /// <summary>
27         /// <para>
28         /// The size.
29         /// </para>
30         /// <para></para>
31         /// </summary>
32         public static readonly long SizeInBytes = Structure<RawLink<TLink>>.Size;
33
34         /// <summary>
35         /// <para>
36         /// The source.
37         /// </para>
38         /// <para></para>
39         /// </summary>
40         public TLink Source;
41         /// <summary>
42         /// <para>
43         /// The target.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         public TLink Target;
48         /// <summary>
49         /// <para>
50         /// The left as source.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         public TLink LeftAsSource;
55         /// <summary>
56         /// <para>
57         /// The right as source.
58         /// </para>
59         /// <para></para>
60         /// </summary>
61         public TLink RightAsSource;
62         /// <summary>
63         /// <para>
64         /// The size as source.
65         /// </para>
66         /// <para></para>
67         /// </summary>
68         public TLink SizeAsSource;
69         /// <summary>
70         /// <para>
71         /// The left as target.
72         /// </para>
73         /// <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);

```

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

1.90 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksRecursionlessSizeBalancedTreeMetho

```
1  using System.Runtime.CompilerServices;
2  using Platform.Data.Doublets.Memory.United.Generic;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.Data.Doublets.Memory.United.Specific
7  {
8      /// <summary>
9      /// <para>
10     /// Represents the int 32 links recursionless size balanced tree methods base.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{uint}"/>
15     public unsafe abstract class UInt32LinksRecursionlessSizeBalancedTreeMethodsBase :
       ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<uint>
16     {
17         /// <summary>
18         /// <para>
19         /// The links.
20         /// </para>
21         /// <para></para>
22         /// </summary>
23         protected new readonly RawLink<uint>* Links;
24         /// <summary>
25         /// <para>
26         /// The header.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         protected new readonly LinksHeader<uint>* Header;
31
32         /// <summary>
33         /// <para>
34         /// Initializes a new <see cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
           ↪ instance.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         /// <param name="constants">
39         /// <para>A constants.</para>
40         /// <para></para>
41         /// </param>
42         /// <param name="links">
43         /// <para>A links.</para>
44         /// <para></para>
45         /// </param>
46         /// <param name="header">
47         /// <para>A header.</para>
48         /// <para></para>
49         /// </param>
50         protected UInt32LinksRecursionlessSizeBalancedTreeMethodsBase(LinksConstants<uint>
           ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header)
51             : base(constants, (byte*)links, (byte*)header)
52         {
53             Links = links;
54             Header = header;
55         }
56
57         /// <summary>
58         /// <para>
59         /// Gets the zero.
60         /// </para>
61         /// <para></para>
62         /// </summary>
63         /// <returns>
64         /// <para>The uint</para>
65         /// <para></para>
66         /// </returns>
67         [MethodImpl(MethodImplOptions.AggressiveInlining)]
68         protected override uint GetZero() => 0U;
69
```

```
        /// <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>
```

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

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

```
        /// <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>
        /// <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>
```

```
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>
```

```
        /// </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">
```

```
343             /// <para>The first.</para>
344             /// <para></para>
345             /// </param>
346             /// <param name="second">
347             /// <para>The second.</para>
348             /// <para></para>
349             /// </param>
350             /// <returns>
351             /// <para>The bool</para>
352             /// <para></para>
353             /// </returns>
354             [MethodImpl(MethodImplOptions.AggressiveInlining)]
355             protected override bool FirstIsToTheLeftOfSecond(uint first, uint second)
356             {
357                 ref var firstLink = ref Links[first];
358                 ref var secondLink = ref Links[second];
359                 return FirstIsToTheLeftOfSecond(firstLink.Source, firstLink.Target,
                    ↪  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(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     }
```

```
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 recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods :
    ↪  UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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 UInt32LinksSourcesRecursionlessSizeBalancedTreeMethods(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>
```

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

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

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

```
        /// 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;
```

```
171
172         /// <summary>
173         /// <para>
174         /// Gets the tree root.
175         /// </para>
176         /// <para></para>
177         /// </summary>
178         /// <returns>
179         /// <para>The uint</para>
180         /// <para></para>
181         /// </returns>
182         [MethodImpl(MethodImplOptions.AggressiveInlining)]
183         protected override uint GetTreeRoot() => Header->RootAsSource;
184
185         /// <summary>
186         /// <para>
187         /// Gets the base part value using the specified link.
188         /// </para>
189         /// <para></para>
190         /// </summary>
191         /// <param name="link">
192         /// <para>The link.</para>
193         /// <para></para>
194         /// </param>
195         /// <returns>
196         /// <para>The uint</para>
197         /// <para></para>
198         /// </returns>
199         [MethodImpl(MethodImplOptions.AggressiveInlining)]
200         protected override uint GetBasePartValue(uint link) => Links[link].Source;
201
202         /// <summary>
203         /// <para>
204         /// Determines whether this instance first is to the left of second.
205         /// </para>
206         /// <para></para>
207         /// </summary>
208         /// <param name="firstSource">
209         /// <para>The first source.</para>
210         /// <para></para>
211         /// </param>
212         /// <param name="firstTarget">
213         /// <para>The first target.</para>
214         /// <para></para>
215         /// </param>
216         /// <param name="secondSource">
217         /// <para>The second source.</para>
218         /// <para></para>
219         /// </param>
220         /// <param name="secondTarget">
221         /// <para>The second target.</para>
222         /// <para></para>
223         /// </param>
224         /// <returns>
225         /// <para>The bool</para>
226         /// <para></para>
227         /// </returns>
228         [MethodImpl(MethodImplOptions.AggressiveInlining)]
229         protected override bool FirstIsToTheLeftOfSecond(uint firstSource, uint firstTarget,
            ↪  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>
248         /// <para></para>
249         /// </param>
250         /// <param name="secondTarget">
251         /// <para>The second target.</para>
252         /// <para></para>
253         /// </param>
254         /// <returns>
255         /// <para>The bool</para>
256         /// <para></para>
257         /// </returns>
258         [MethodImpl(MethodImplOptions.AggressiveInlining)]
259         protected override bool FirstIsToTheRightOfSecond(uint firstSource, uint firstTarget,
            ↪ uint secondSource, uint secondTarget)
260             => firstSource > secondSource || (firstSource == secondSource && firstTarget >
                ↪ secondTarget);
261 
262         /// <summary>
263         /// <para>
264         /// Clears the node using the specified node.
265         /// </para>
266         /// <para></para>
267         /// </summary>
268         /// <param name="node">
269         /// <para>The node.</para>
270         /// <para></para>
271         /// </param>
272         [MethodImpl(MethodImplOptions.AggressiveInlining)]
273         protected override void ClearNode(uint node)
274         {
275             ref var link = ref Links[node];
276             link.LeftAsSource = 0U;
277             link.RightAsSource = 0U;
278             link.SizeAsSource = 0U;
279         }
280     }
281 }
```

1.94 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32LinksTargetsRecursionlessSizeBalancedTre

```
1  using System.Runtime.CompilerServices;
2  
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4  
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 32 links targets recursionless size balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt32LinksRecursionlessSizeBalancedTreeMethodsBase"/>
14     public unsafe class UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods :
        ↪ UInt32LinksRecursionlessSizeBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see
            ↪ cref="UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods"/> instance.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="constants">
23         /// <para>A constants.</para>
24         /// <para></para>
25         /// </param>
26         /// <param name="links">
27         /// <para>A links.</para>
28         /// <para></para>
29         /// </param>
30         /// <param name="header">
31         /// <para>A header.</para>
32         /// <para></para>
33         /// </param>
34         public UInt32LinksTargetsRecursionlessSizeBalancedTreeMethods(LinksConstants<uint>
            ↪ constants, RawLink<uint>* links, LinksHeader<uint>* header) : base(constants, links,
            ↪ header) { }
35 
```

```
        /// <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)]
        protected override bool FirstIsToTheLeftOfSecond(uint firstSource, uint firstTarget,
        ↪  uint secondSource, uint 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(uint firstSource, uint firstTarget,
        ↪  uint secondSource, uint 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(uint node)
        {
            ref var link = ref Links[node];
            link.LeftAsTarget = 0U;
            link.RightAsTarget = 0U;
            link.SizeAsTarget = 0U;
        }
    }
}
```

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>
```

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

```
        /// <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)
            => 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(uint firstSource, uint firstTarget,
        ↪  uint secondSource, uint 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(uint node)
        {
            ref var link = ref Links[node];
            link.LeftAsTarget = OU;
            link.RightAsTarget = OU;
            link.SizeAsTarget = OU;
        }
    }
}
```

```
using System;
using System.Runtime.CompilerServices;
using Platform.Memory;
using Platform.Singletons;
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 a low-level implementation of direct access to resizable memory, for organizing the storage of links with addresses represented as <see cref="uint" />.</para>
    /// <para>Представляет низкоуровневую реализация прямого доступа к памяти с переменным размером, для организации хранения связей с адресами представленными в виде <see cref="uint"/>.</para>
    /// </summary>
    public unsafe class UInt32UnitedMemoryLinks : UnitedMemoryLinksBase<uint>
    {
        /// <summary>
        /// <para>
        /// The create source tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<uint>> _createSourceTreeMethods;
        /// <summary>
        /// <para>
        /// The create target tree methods.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Func<ILinksTreeMethods<uint>> _createTargetTreeMethods;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        private LinksHeader<uint>* _header;
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        private RawLink<uint>* _links;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="address">
        /// <para>A address.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public UInt32UnitedMemoryLinks(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)]
        public UInt32UnitedMemoryLinks(string address, long memoryReservationStep) : this(new FileMappedResizableDirectMemory(address, memoryReservationStep), memoryReservationStep) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt32UnitedMemoryLinks"/> instance.
        /// </para>
```

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

```
137         /// Sets the pointers using the specified memory.
138         /// </para>
139         /// <para></para>
140         /// </summary>
141         /// <param name="memory">
142         /// <para>The memory.</para>
143         /// <para></para>
144         /// </param>
145         [MethodImpl(MethodImplOptions.AggressiveInlining)]
146         protected override void SetPointers(IResizableDirectMemory memory)
147         {
148             _header = (LinksHeader<uint>*)memory.Pointer;
149             _links = (RawLink<uint>*)memory.Pointer;
150             SourcesTreeMethods = _createSourceTreeMethods();
151             TargetsTreeMethods = _createTargetTreeMethods();
152             UnusedLinksListMethods = new UInt32UnusedLinksListMethods(_links, _header);
153         }
154 
155         /// <summary>
156         /// <para>
157         /// Resets the pointers.
158         /// </para>
159         /// <para></para>
160         /// </summary>
161         [MethodImpl(MethodImplOptions.AggressiveInlining)]
162         protected override void ResetPointers()
163         {
164             base.ResetPointers();
165             _links = null;
166             _header = null;
167         }
168 
169         /// <summary>
170         /// <para>
171         /// Gets the header reference.
172         /// </para>
173         /// <para></para>
174         /// </summary>
175         /// <returns>
176         /// <para>A ref links header of uint</para>
177         /// <para></para>
178         /// </returns>
179         [MethodImpl(MethodImplOptions.AggressiveInlining)]
180         protected override ref LinksHeader<uint> GetHeaderReference() => ref *_header;
181 
182         /// <summary>
183         /// <para>
184         /// Gets the link reference using the specified link index.
185         /// </para>
186         /// <para></para>
187         /// </summary>
188         /// <param name="linkIndex">
189         /// <para>The link index.</para>
190         /// <para></para>
191         /// </param>
192         /// <returns>
193         /// <para>A ref raw link of uint</para>
194         /// <para></para>
195         /// </returns>
196         [MethodImpl(MethodImplOptions.AggressiveInlining)]
197         protected override ref RawLink<uint> GetLinkReference(uint linkIndex) => ref
            ↪ _links[linkIndex];
198 
199         /// <summary>
200         /// <para>
201         /// Determines whether this instance are equal.
202         /// </para>
203         /// <para></para>
204         /// </summary>
205         /// <param name="first">
206         /// <para>The first.</para>
207         /// <para></para>
208         /// </param>
209         /// <param name="second">
210         /// <para>The second.</para>
211         /// <para></para>
212         /// </param>
213         /// <returns>
```

```
        /// <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>
```

```
370         /// <param name="first">
371         /// <para>The first.</para>
372         /// <para></para>
373         /// </param>
374         /// <param name="second">
375         /// <para>The second.</para>
376         /// <para></para>
377         /// </param>
378         /// <returns>
379         /// <para>The uint</para>
380         /// <para></para>
381         /// </returns>
382         [MethodImpl(MethodImplOptions.AggressiveInlining)]
383         protected override uint Add(uint first, uint second) => first + second;
384 
385         /// <summary>
386         /// <para>
387         /// Subtracts the first.
388         /// </para>
389         /// <para></para>
390         /// </summary>
391         /// <param name="first">
392         /// <para>The first.</para>
393         /// <para></para>
394         /// </param>
395         /// <param name="second">
396         /// <para>The second.</para>
397         /// <para></para>
398         /// </param>
399         /// <returns>
400         /// <para>The uint</para>
401         /// <para></para>
402         /// </returns>
403         [MethodImpl(MethodImplOptions.AggressiveInlining)]
404         protected override uint Subtract(uint first, uint second) => first - second;
405 
406         /// <summary>
407         /// <para>
408         /// Increments the link.
409         /// </para>
410         /// <para></para>
411         /// </summary>
412         /// <param name="link">
413         /// <para>The link.</para>
414         /// <para></para>
415         /// </param>
416         /// <returns>
417         /// <para>The uint</para>
418         /// <para></para>
419         /// </returns>
420         [MethodImpl(MethodImplOptions.AggressiveInlining)]
421         protected override uint Increment(uint link) => ++link;
422 
423         /// <summary>
424         /// <para>
425         /// Decrements the link.
426         /// </para>
427         /// <para></para>
428         /// </summary>
429         /// <param name="link">
430         /// <para>The link.</para>
431         /// <para></para>
432         /// </param>
433         /// <returns>
434         /// <para>The uint</para>
435         /// <para></para>
436         /// </returns>
437         [MethodImpl(MethodImplOptions.AggressiveInlining)]
438         protected override uint Decrement(uint link) => --link;
439     }
440 }
```

1.97 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt32UnusedLinksListMethods.cs

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

```
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>
    {
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly RawLink<uint>* _links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        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>
```

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

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

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

```
        {
            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 size value 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 GetSizeValue(ulong value) => (value & 4294967264UL) >> 5;

        /// <summary>
        /// <para>
        /// Sets the size value using the specified stored value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storedValue">
        /// <para>The stored value.</para>
        /// <para></para>
        /// </param>
        /// <param name="size">
        /// <para>The size.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override void SetSizeValue(ref ulong storedValue, ulong size) => storedValue =
        ↪  storedValue & 31UL | (size & 134217727UL) << 5;

        /// <summary>
        /// <para>
        /// Determines whether this instance get left 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 override bool GetLeftIsChildValue(ulong value) => (value & 16UL) >> 4 == 1UL;

        /// <summary>
        /// <para>
        /// 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 override void SetLeftIsChildValue(ref ulong storedValue, bool value) =>
        ↪  storedValue = storedValue & 4294967279UL | (As<bool, byte>(ref value) & 1UL) << 4;

        /// <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 override bool GetRightIsChildValue(ulong value) => (value & 8UL) >> 3 == 1UL;

        /// <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 override void SetRightIsChildValue(ref ulong storedValue, bool value) =>
        ↪  storedValue = storedValue & 4294967287UL | (As<bool, byte>(ref value) & 1UL) << 3;

        /// <summary>
        /// <para>
        /// Gets the balance value using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The sbyte</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override sbyte GetBalanceValue(ulong value) => unchecked((sbyte)(value & 7UL |
        ↪  0xF8UL * ((value & 4UL) >> 2))); // if negative, then continue ones to the end of
        ↪  sbyte

        /// <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 override void SetBalanceValue(ref ulong storedValue, sbyte value) =>
        ↪  storedValue = unchecked(storedValue & 4294967288UL | (ulong)((byte)value >> 5 & 4 |
        ↪  value & 3) & 7UL);

        /// <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>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
    }
}
```

1.99 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksRecursionlessSizeBalancedTreeMetho

```
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 64 links recursionless size balanced tree methods base.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksRecursionlessSizeBalancedTreeMethodsBase{ulong}"/>
    public unsafe abstract class UInt64LinksRecursionlessSizeBalancedTreeMethodsBase :
    ↪ LinksRecursionlessSizeBalancedTreeMethodsBase<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="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
        ↪ 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>
```

```
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
184         /// <summary>
185         /// <para>
186         /// Determines whether this instance less or equal than zero.
187         /// </para>
188         /// <para></para>
189         /// </summary>
190         /// <param name="value">
191         /// <para>The value.</para>
192         /// <para></para>
193         /// </param>
194         /// <returns>
195         /// <para>The bool</para>
196         /// <para></para>
197         /// </returns>
198         [MethodImpl(MethodImplOptions.AggressiveInlining)]
199         protected override bool LessOrEqualThanZero(ulong value) => value == 0UL; // value is
            ↪  always >= 0 for ulong
200
201         /// <summary>
202         /// <para>
```

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

```
        /// </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>
        /// 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];
```

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

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

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

```
14      /// <seealso cref="LinksSizeBalancedTreeMethodsBase{ulong}"/>
15      public unsafe abstract class UInt64LinksSizeBalancedTreeMethodsBase :
        ↪ LinksSizeBalancedTreeMethodsBase<ulong>
16      {
17          /// <summary>
18          /// <para>
19          /// The links.
20          /// </para>
21          /// <para></para>
22          /// </summary>
23          protected new readonly RawLink<ulong>* Links;
24          /// <summary>
25          /// <para>
26          /// The header.
27          /// </para>
28          /// <para></para>
29          /// </summary>
30          protected new readonly LinksHeader<ulong>* Header;
31
32          /// <summary>
33          /// <para>
34          /// Initializes a new <see cref="UInt64LinksSizeBalancedTreeMethodsBase"/> instance.
35          /// </para>
36          /// <para></para>
37          /// </summary>
38          /// <param name="constants">
39          /// <para>A constants.</para>
40          /// <para></para>
41          /// </param>
42          /// <param name="links">
43          /// <para>A links.</para>
44          /// <para></para>
45          /// </param>
46          /// <param name="header">
47          /// <para>A header.</para>
48          /// <para></para>
49          /// </param>
50          protected UInt64LinksSizeBalancedTreeMethodsBase(LinksConstants<ulong> constants,
            ↪ RawLink<ulong>* links, LinksHeader<ulong>* header)
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.
```

```
        /// </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>
        /// <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>
```

```
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>
```

```
        /// </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>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override ref RawLink<ulong> GetLinkReference(ulong link) => ref Links[link];
    }
}
```

1.101 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesAvlBalancedTreeMethods.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 sources avl balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
    public unsafe class UInt64LinksSourcesAvlBalancedTreeMethods :
    ↪ UInt64LinksAvlBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64LinksSourcesAvlBalancedTreeMethods"/> 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 UInt64LinksSourcesAvlBalancedTreeMethods(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)]
```

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

```
126         /// </summary>
127         /// <param name="node">
128         /// <para>The node.</para>
129         /// <para></para>
130         /// </param>
131         /// <param name="right">
132         /// <para>The right.</para>
133         /// <para></para>
134         /// </param>
135         [MethodImpl(MethodImplOptions.AggressiveInlining)]
136         protected override void SetRight(ulong node, ulong right) => Links[node].RightAsSource =
            ↪ right;
137
138         /// <summary>
139         /// <para>
140         /// Gets the size using the specified node.
141         /// </para>
142         /// <para></para>
143         /// </summary>
144         /// <param name="node">
145         /// <para>The node.</para>
146         /// <para></para>
147         /// </param>
148         /// <returns>
149         /// <para>The ulong</para>
150         /// <para></para>
151         /// </returns>
152         [MethodImpl(MethodImplOptions.AggressiveInlining)]
153         protected override ulong GetSize(ulong node) => GetSizeValue(Links[node].SizeAsSource);
154
155         /// <summary>
156         /// <para>
157         /// Sets the size using the specified node.
158         /// </para>
159         /// <para></para>
160         /// </summary>
161         /// <param name="node">
162         /// <para>The node.</para>
163         /// <para></para>
164         /// </param>
165         /// <param name="size">
166         /// <para>The size.</para>
167         /// <para></para>
168         /// </param>
169         [MethodImpl(MethodImplOptions.AggressiveInlining)]
170         protected override void SetSize(ulong node, ulong size) => SetSizeValue(ref
            ↪ Links[node].SizeAsSource, size);
171
172         /// <summary>
173         /// <para>
174         /// Determines whether this instance get left is child.
175         /// </para>
176         /// <para></para>
177         /// </summary>
178         /// <param name="node">
179         /// <para>The node.</para>
180         /// <para></para>
181         /// </param>
182         /// <returns>
183         /// <para>The bool</para>
184         /// <para></para>
185         /// </returns>
186         [MethodImpl(MethodImplOptions.AggressiveInlining)]
187         protected override bool GetLeftIsChild(ulong node) =>
            ↪ 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>
257             /// <para>The sbyte</para>
258             /// <para></para>
259             /// </returns>
260             [MethodImpl(MethodImplOptions.AggressiveInlining)]
261             protected override sbyte GetBalance(ulong node) =>
                ↪ GetBalanceValue(Links[node].SizeAsSource);
262
263             /// <summary>
264             /// <para>
265             /// Sets the balance using the specified node.
266             /// </para>
267             /// <para></para>
268             /// </summary>
269             /// <param name="node">
270             /// <para>The node.</para>
271             /// <para></para>
272             /// </param>
273             /// <param name="value">
274             /// <para>The value.</para>
```

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

```
        /// <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;
        }
    }
}
```

1.102 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksSourcesRecursionlessSizeBalancedTr

```
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 recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods :
    ↪  UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods"/> 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>
```

```
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.
```

```
        /// </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) => 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) => Links[node].SizeAsSource =
        ↪  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->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;
        }
    }
}
```

1.103 ./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;

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

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

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

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

```
1  using System.Runtime.CompilerServices;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Memory.United.Specific
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the int 64 links targets avl balanced tree methods.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="UInt64LinksAvlBalancedTreeMethodsBase"/>
14     public unsafe class UInt64LinksTargetsAvlBalancedTreeMethods :
       ↪  UInt64LinksAvlBalancedTreeMethodsBase
15     {
16         /// <summary>
17         /// <para>
18         /// Initializes a new <see cref="UInt64LinksTargetsAvlBalancedTreeMethods"/> 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 UInt64LinksTargetsAvlBalancedTreeMethods(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>
```

```
        /// <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>
        /// 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) => GetSizeValue(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) => SetSizeValue(ref
        ↪  Links[node].SizeAsTarget, 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>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GetLeftIsChild(ulong node) =>
        ↪  GetLeftIsChildValue(Links[node].SizeAsTarget);

        /// <summary>
        /// <para>
        /// Sets the left is child 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 SetLeftIsChild(ulong node, bool value) =>
        ↪  SetLeftIsChildValue(ref Links[node].SizeAsTarget, value);

        /// <summary>
        /// <para>
        /// Determines whether this instance get right is child.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="node">
```

```
        /// <para>The node.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool GetRightIsChild(ulong node) =>
        ↪  GetRightIsChildValue(Links[node].SizeAsTarget);

        /// <summary>
        /// <para>
        /// Sets the right is child 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 SetRightIsChild(ulong node, bool value) =>
        ↪  SetRightIsChildValue(ref Links[node].SizeAsTarget, value);

        /// <summary>
        /// <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(ulong node) =>
        ↪  GetBalanceValue(Links[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(ulong node, sbyte value) => SetBalanceValue(ref
        ↪  Links[node].SizeAsTarget, 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->RootAsTarget;

```

```
287         /// <summary>
288         /// <para>
289         /// Gets the base part value using the specified link.
290         /// </para>
291         /// <para></para>
292         /// </summary>
293         /// <param name="link">
294         /// <para>The link.</para>
295         /// <para></para>
296         /// </param>
297         /// <returns>
298         /// <para>The ulong</para>
299         /// <para></para>
300         /// </returns>
301         [MethodImpl(MethodImplOptions.AggressiveInlining)]
302         protected override ulong GetBasePartValue(ulong link) => Links[link].Target;
303
304         /// <summary>
305         /// <para>
306         /// Determines whether this instance first is to the left of second.
307         /// </para>
308         /// <para></para>
309         /// </summary>
310         /// <param name="firstSource">
311         /// <para>The first source.</para>
312         /// <para></para>
313         /// </param>
314         /// <param name="firstTarget">
315         /// <para>The first target.</para>
316         /// <para></para>
317         /// </param>
318         /// <param name="secondSource">
319         /// <para>The second source.</para>
320         /// <para></para>
321         /// </param>
322         /// <param name="secondTarget">
323         /// <para>The second target.</para>
324         /// <para></para>
325         /// </param>
326         /// <returns>
327         /// <para>The bool</para>
328         /// <para></para>
329         /// </returns>
330         [MethodImpl(MethodImplOptions.AggressiveInlining)]
331         protected override bool FirstIsToTheLeftOfSecond(ulong firstSource, ulong firstTarget,
          ↪  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)
```

```
            => 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(ulong node)
        {
            ref var link = ref Links[node];
            link.LeftAsTarget = OUL;
            link.RightAsTarget = OUL;
            link.SizeAsTarget = OUL;
        }
    }
}
```

1.105 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsRecursionlessSizeBalancedTr

```
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 recursionless size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64LinksRecursionlessSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods :
    ↪  UInt64LinksRecursionlessSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see
        ↪  cref="UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods"/> 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 UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(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>
```

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

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

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

```
        {
            ref var link = ref Links[node];
            link.LeftAsTarget = OUL;
            link.RightAsTarget = OUL;
            link.SizeAsTarget = OUL;
        }
    }
}
```

1.106 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64LinksTargetsSizeBalancedTreeMethods.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 size balanced tree methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UInt64LinksSizeBalancedTreeMethodsBase"/>
    public unsafe class UInt64LinksTargetsSizeBalancedTreeMethods :
    ↪ UInt64LinksSizeBalancedTreeMethodsBase
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64LinksTargetsSizeBalancedTreeMethods"/> 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 UInt64LinksTargetsSizeBalancedTreeMethods(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].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>
```

```
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>
```

```
        /// 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>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        protected override bool FirstIsToTheRightOfSecond(ulong firstSource, ulong firstTarget,
        ↪  ulong secondSource, ulong 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(ulong node)
        {
            ref var link = ref Links[node];
            link.LeftAsTarget = OUL;
            link.RightAsTarget = OUL;
            link.SizeAsTarget = OUL;
        }
    }
}
```

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

```
using System;
using System.Runtime.CompilerServices;
using Platform.Memory;
using Platform.Singletons;
using Platform.Data.Doublets.Memory.United.Generic;

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

```
78          public UInt64UnitedMemoryLinks(IResizableDirectMemory memory) : this(memory,
            ↪  DefaultLinksSizeStep) { }
79
80          /// <summary>
81          /// <para>
82          /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
83          /// </para>
84          /// <para></para>
85          /// </summary>
86          /// <param name="memory">
87          /// <para>A memory.</para>
88          /// <para></para>
89          /// </param>
90          /// <param name="memoryReservationStep">
91          /// <para>A memory reservation step.</para>
92          /// <para></para>
93          /// </param>
94          [MethodImpl(MethodImplOptions.AggressiveInlining)]
95          public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
            ↪  memoryReservationStep) : this(memory, memoryReservationStep,
            ↪  Default<LinksConstants<ulong>>.Instance, IndexTreeType.Default) { }
96
97          /// <summary>
98          /// <para>
99          /// Initializes a new <see cref="UInt64UnitedMemoryLinks"/> instance.
100         /// </para>
101         /// <para></para>
102         /// </summary>
103         /// <param name="memory">
104         /// <para>A memory.</para>
105         /// <para></para>
106         /// </param>
107         /// <param name="memoryReservationStep">
108         /// <para>A memory reservation step.</para>
109         /// <para></para>
110         /// </param>
111         /// <param name="constants">
112         /// <para>A constants.</para>
113         /// <para></para>
114         /// </param>
115         /// <param name="indexTreeType">
116         /// <para>A index tree type.</para>
117         /// <para></para>
118         /// </param>
119         [MethodImpl(MethodImplOptions.AggressiveInlining)]
120         public UInt64UnitedMemoryLinks(IResizableDirectMemory memory, long
            ↪  memoryReservationStep, LinksConstants<ulong> constants, IndexTreeType indexTreeType)
            ↪  : base(memory, memoryReservationStep, constants)
121         {
122             if (indexTreeType == IndexTreeType.SizedAndThreadedAVLBalancedTree)
123             {
124                 _createSourceTreeMethods = () => new
                    ↪  UInt64LinksSourcesAvlBalancedTreeMethods(Constants, _links, _header);
125                 _createTargetTreeMethods = () => new
                    ↪  UInt64LinksTargetsAvlBalancedTreeMethods(Constants, _links, _header);
126             }
127             else if (indexTreeType == IndexTreeType.SizeBalancedTree)
128             {
129                 _createSourceTreeMethods = () => new
                    ↪  UInt64LinksSourcesSizeBalancedTreeMethods(Constants, _links, _header);
130                 _createTargetTreeMethods = () => new
                    ↪  UInt64LinksTargetsSizeBalancedTreeMethods(Constants, _links, _header);
131             }
132             else
133             {
134                 _createSourceTreeMethods = () => new
                    ↪  UInt64LinksSourcesRecursionlessSizeBalancedTreeMethods(Constants, _links,
                    ↪  _header);
135                 _createTargetTreeMethods = () => new
                    ↪  UInt64LinksTargetsRecursionlessSizeBalancedTreeMethods(Constants, _links,
                    ↪  _header);
136             }
137             Init(memory, memoryReservationStep);
138         }
139
140         /// <summary>
141         /// <para>
```

```
142         /// Sets the pointers using the specified memory.
143         /// </para>
144         /// <para></para>
145         /// </summary>
146         /// <param name="memory">
147         /// <para>The memory.</para>
148         /// <para></para>
149         /// </param>
150         [MethodImpl(MethodImplOptions.AggressiveInlining)]
151         protected override void SetPointers(IResizableDirectMemory memory)
152         {
153             _header = (LinksHeader<ulong>*)memory.Pointer;
154             _links = (RawLink<ulong>*)memory.Pointer;
155             SourcesTreeMethods = _createSourceTreeMethods();
156             TargetsTreeMethods = _createTargetTreeMethods();
157             UnusedLinksListMethods = new UInt64UnusedLinksListMethods(_links, _header);
158         }
159 
160         /// <summary>
161         /// <para>
162         /// Resets the pointers.
163         /// </para>
164         /// <para></para>
165         /// </summary>
166         [MethodImpl(MethodImplOptions.AggressiveInlining)]
167         protected override void ResetPointers()
168         {
169             base.ResetPointers();
170             _links = null;
171             _header = null;
172         }
173 
174         /// <summary>
175         /// <para>
176         /// Gets the header reference.
177         /// </para>
178         /// <para></para>
179         /// </summary>
180         /// <returns>
181         /// <para>A ref links header of ulong</para>
182         /// <para></para>
183         /// </returns>
184         [MethodImpl(MethodImplOptions.AggressiveInlining)]
185         protected override ref LinksHeader<ulong> GetHeaderReference() => ref *_header;
186 
187         /// <summary>
188         /// <para>
189         /// Gets the link reference using the specified link index.
190         /// </para>
191         /// <para></para>
192         /// </summary>
193         /// <param name="linkIndex">
194         /// <para>The link index.</para>
195         /// <para></para>
196         /// </param>
197         /// <returns>
198         /// <para>A ref raw link of ulong</para>
199         /// <para></para>
200         /// </returns>
201         [MethodImpl(MethodImplOptions.AggressiveInlining)]
202         protected override ref RawLink<ulong> GetLinkReference(ulong linkIndex) => ref
            ↪  _links[linkIndex];
203 
204         /// <summary>
205         /// <para>
206         /// Determines whether this instance are equal.
207         /// </para>
208         /// <para></para>
209         /// </summary>
210         /// <param name="first">
211         /// <para>The first.</para>
212         /// <para></para>
213         /// </param>
214         /// <param name="second">
215         /// <para>The second.</para>
216         /// <para></para>
217         /// </param>
218         /// <returns>
```

```
        /// <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>
```

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

1.108 ./csharp/Platform.Data.Doublets/Memory/United/Specific/UInt64UnusedLinksListMethods.cs

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

```
namespace Platform.Data.Doublets.Memory.United.Specific
{
    /// <summary>
    /// <para>
    /// Represents the int 64 unused links list methods.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="UnusedLinksListMethods{ulong}"/>
    public unsafe class UInt64UnusedLinksListMethods : UnusedLinksListMethods<ulong>
    {
        /// <summary>
        /// <para>
        /// The links.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly RawLink<ulong>* _links;
        /// <summary>
        /// <para>
        /// The header.
        /// </para>
        /// <para></para>
        /// </summary>
        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}"/>
    public class PropertiesOperator<TLink> : LinksOperatorBase<TLink>, IProperties<TLink, TLink,
    ↪  TLink>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

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

        /// <summary>
        /// <para>
        /// Gets the value using the specified object.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@object">
        /// <para>The object.</para>
        /// <para></para>
        /// </param>
        /// <param name="property">
        /// <para>The property.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink GetValue(TLink @object, TLink property)
        {
            var links = _links;
            var objectProperty = links.SearchOrDefault(@object, property);
            if (_equalityComparer.Equals(objectProperty, default))
            {
                return default;
            }
            var constants = links.Constants;
            var any = constants.Any;
            var query = new Link<TLink>(any, objectProperty, any);
            var valueLink = links.SingleOrDefault(query);
            if (valueLink == null)
            {
                return default;
            }
```

```
            return links.GetTarget(valueLink[constants.IndexPart]);
        }

        /// <summary>
        /// <para>
        /// Sets the value using the specified object.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@object">
        /// <para>The object.</para>
        /// <para></para>
        /// </param>
        /// <param name="property">
        /// <para>The property.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void SetValue(TLink @object, TLink property, TLink value)
        {
            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>
    {
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// The property marker.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly TLink _propertyMarker;
        /// <summary>
        /// <para>
        /// The property value marker.
        /// </para>
        /// <para></para>
        /// </summary>
        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));
        }

        /// <summary>
        /// <para>
        /// Gets the container using the specified property.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="property">
        /// <para>The property.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The value container.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private TLink GetContainer(TLink property)
        {
            var valueContainer = default(TLink);
            if (_equalityComparer.Equals(property, default))
            {
                return valueContainer;
            }
            var links = _links;
            var constants = links.Constants;
            var countinueConstant = constants.Continue;
            var breakConstant = constants.Break;
            var anyConstant = constants.Any;
            var query = new Link<TLink>(anyConstant, property, anyConstant);
            links.Each(candidate =>
            {
                var candidateTarget = links.GetTarget(candidate);
                var valueTarget = links.GetTarget(candidateTarget);
                if (_equalityComparer.Equals(valueTarget, _propertyValueMarker))
                {
                    valueContainer = links.GetIndex(candidate);
```

```
                    return breakConstant;
                }
                return countinueConstant;
            }, query);
            return valueContainer;
        }

        /// <summary>
        /// <para>
        /// Gets the value using the specified container.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="container">
        /// <para>The container.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private TLink GetValue(TLink container) => _equalityComparer.Equals(container, default)
        ↪  ? default : _links.GetTarget(container);

        /// <summary>
        /// <para>
        /// Sets the link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Set(TLink link, TLink value)
        {
            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>
    {
        /// <summary>
        /// <para>
        /// The default.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly EqualityComparer<TLink> _equalityComparer =
        ↪  EqualityComparer<TLink>.Default;

        /// <summary>
        /// <para>
        /// The stack.
        /// </para>
        /// <para></para>
        /// </summary>
        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)]
        private TLink GetStackMarker() => _links.GetSource(_stack);

        /// <summary>
        /// <para>
        /// Gets the top.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private TLink GetTop() => _links.GetTarget(_stack);

        /// <summary>
        /// <para>
        /// Peeks this instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
```

```
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink Peek() => _links.GetTarget(GetTop());

        /// <summary>
        /// <para>
        /// Pops this instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The element.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLink Pop()
        {
            var element = Peek();
            if (!_equalityComparer.Equals(element, _stack))
            {
                var top = GetTop();
                var previousTop = _links.GetSource(top);
                _links.Update(_stack, GetStackMarker(), previousTop);
                _links.Delete(top);
            }
            return element;
        }

        /// <summary>
        /// <para>
        /// Pushes the element.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="element">
        /// <para>The element.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void Push(TLink element) => _links.Update(_stack, GetStackMarker(),
        ↪  _links.GetOrCreate(GetTop(), element));
    }
}
```

1.112 ./csharp/Platform.Data.Doublets/Stacks/StackExtensions.cs

```
using System.Runtime.CompilerServices;

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

namespace Platform.Data.Doublets.Stacks
{
    /// <summary>
    /// <para>
    /// 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);

        /// <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 address</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public TLinkAddress Each(Func<IList<TLinkAddress>, TLinkAddress> handler,
        ↪  IList<TLinkAddress> restrictions) => SyncRoot.ExecuteReadOperation(handler,
        ↪  restrictions, (handler1, restrictions1) => Unsync.Each(handler1, restrictions1));

        /// <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
```

```
{
    /// <summary>
    /// <para>
    /// Represents the int 64 links extensions.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class UInt64LinksExtensions
    {
        /// <summary>
        /// <para>
        /// The instance.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly LinksConstants<ulong> Constants =
        ↪  Default<LinksConstants<ulong>>.Instance;

        /// <summary>
        /// <para>
        /// Determines whether any link is any.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="sequence">
        /// <para>The sequence.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The bool</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool AnyLinkIsAny(this ILinks<ulong> links, params ulong[] sequence)
        {
            if (sequence == null)
            {
                return false;
            }
            var constants = links.Constants;
            for (var i = 0; i < sequence.Length; i++)
            {
                if (sequence[i] == constants.Any)
                {
                    return true;
                }
            }
            return false;
        }

        /// <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>
```

```
87          /// <para></para>
88          /// </param>
89          /// <returns>
90          /// <para>The string</para>
91          /// <para></para>
92          /// </returns>
93          [MethodImpl(MethodImplOptions.AggressiveInlining)]
94          public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
            ↪  Func<Link<ulong>, bool> isElement, bool renderIndex = false, bool renderDebug =
            ↪  false)
95          {
96              var sb = new StringBuilder();
97              var visited = new HashSet<ulong>();
98              links.AppendStructure(sb, visited, linkIndex, isElement, (innerSb, link) =>
                ↪  innerSb.Append(link.Index), renderIndex, renderDebug);
99              return sb.ToString();
100         }
101
102         /// <summary>
103         /// <para>
104         /// Formats the structure using the specified links.
105         /// </para>
106         /// <para></para>
107         /// </summary>
108         /// <param name="links">
109         /// <para>The links.</para>
110         /// <para></para>
111         /// </param>
112         /// <param name="linkIndex">
113         /// <para>The link index.</para>
114         /// <para></para>
115         /// </param>
116         /// <param name="isElement">
117         /// <para>The is element.</para>
118         /// <para></para>
119         /// </param>
120         /// <param name="appendElement">
121         /// <para>The append element.</para>
122         /// <para></para>
123         /// </param>
124         /// <param name="renderIndex">
125         /// <para>The render index.</para>
126         /// <para></para>
127         /// </param>
128         /// <param name="renderDebug">
129         /// <para>The render debug.</para>
130         /// <para></para>
131         /// </param>
132         /// <returns>
133         /// <para>The string</para>
134         /// <para></para>
135         /// </returns>
136         [MethodImpl(MethodImplOptions.AggressiveInlining)]
137         public static string FormatStructure(this ILinks<ulong> links, ulong linkIndex,
            ↪  Func<Link<ulong>, bool> isElement, Action<StringBuilder, Link<ulong>> appendElement,
            ↪  bool renderIndex = false, bool renderDebug = false)
138         {
139             var sb = new StringBuilder();
140             var visited = new HashSet<ulong>();
141             links.AppendStructure(sb, visited, linkIndex, isElement, appendElement, renderIndex,
                ↪  renderDebug);
142             return sb.ToString();
143         }
144
145         /// <summary>
146         /// <para>
147         /// Appends the structure using the specified links.
148         /// </para>
149         /// <para></para>
150         /// </summary>
151         /// <param name="links">
152         /// <para>The links.</para>
153         /// <para></para>
154         /// </param>
155         /// <param name="sb">
156         /// <para>The sb.</para>
157         /// <para></para>
```

```
        /// </param>
        /// <param name="visited">
        /// <para>The visited.</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>
        /// <exception cref="ArgumentNullException">
        /// <para></para>
        /// <para></para>
        /// </exception>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        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)
        {
            if (sb == null)
            {
                throw new ArgumentNullException(nameof(sb));
            }
            if (linkIndex == Constants.Null || linkIndex == Constants.Any || linkIndex ==
            ↪  Constants.Itself)
            {
                return;
            }
            if (links.Exists(linkIndex))
            {
                if (visited.Add(linkIndex))
                {
                    sb.Append('(');
                    var link = new Link<ulong>(links.GetLink(linkIndex));
                    if (renderIndex)
                    {
                        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
{
    /// <summary>
    /// <para>
    /// Represents the int 64 links transactions layer.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="LinksDisposableDecoratorBase{ulong}"/>
    public class UInt64LinksTransactionsLayer : LinksDisposableDecoratorBase<ulong> //-V3073
    {
        /// <remarks>
        /// Альтернативные варианты хранения трансформации (элемента транзакции):
        ///
        /// private enum TransitionType
        /// {
        ///     Creation,
        ///     UpdateOf,
        ///     UpdateTo,
        ///     Deletion
        /// }
        ///
        /// private struct Transition
        /// {
        ///     public ulong TransactionId;
        ///     public UniqueTimestamp Timestamp;
        ///     public TransactionItemType Type;
```

```
        ///     public Link Source;
        ///     public Link Linker;
        ///     public Link Target;
        /// }
        ///
        /// Или
        ///
        /// public struct TransitionHeader
        /// {
        ///     public ulong TransactionIdCombined;
        ///     public ulong TimestampCombined;
        ///
        ///     public ulong TransactionId
        ///     {
        ///         get
        ///         {
        ///             return (ulong) mask &amp;amp; TransactionIdCombined;
        ///         }
        ///     }
        ///
        ///     public UniqueTimestamp Timestamp
        ///     {
        ///         get
        ///         {
        ///             return (UniqueTimestamp)mask &amp;amp; TransactionIdCombined;
        ///         }
        ///     }
        ///
        ///     public TransactionItemType Type
        ///     {
        ///         get
        ///         {
        ///             // Использовать по одному биту из TransactionId и Timestamp,
        ///             // для значения в 2 бита, которое представляет тип операции
        ///             throw new NotImplementedException();
        ///         }
        ///     }
        /// }
        ///
        /// private struct Transition
        /// {
        ///     public TransitionHeader Header;
        ///     public Link Source;
        ///     public Link Linker;
        ///     public Link Target;
        /// }
        ///
        /// </remarks>
        public struct Transition : IEquatable<Transition>
        {
            /// <summary>
            /// <para>
            /// 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>
        /// <para></para>
        /// </param>
        /// <param name="before">
        /// <para>A before.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Transition(UniqueTimestampFactory uniqueTimestampFactory, ulong
        ↪  transactionId, Link<ulong> before) : this(uniqueTimestampFactory, transactionId,
        ↪  before, default) { }

        /// <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>
```

```
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             /// <summary>
299             /// <para>
300             /// The transitions.
301             /// </para>
302             /// <para></para>
303             /// </summary>
304             private readonly Queue<Transition> _transitions;
305             /// <summary>
306             /// <para>
307             /// The layer.
308             /// </para>
309             /// <para></para>
310             /// </summary>
311             private readonly UInt64LinksTransactionsLayer _layer;
312             /// <summary>
313             /// <para>
314             /// Gets or sets the is committed value.
315             /// </para>
316             /// <para></para>
317             /// </summary>
318             public bool IsCommitted { get; private set; }
319             /// <summary>
320             /// <para>
321             /// Gets or sets the is reverted value.
322             /// </para>
323             /// <para></para>
324             /// </summary>
325             public bool IsReverted { get; private set; }
326
327             /// <summary>
328             /// <para>
329             /// Initializes a new <see cref="Transaction"/> instance.
330             /// </para>
331             /// <para></para>
332             /// </summary>
333             /// <param name="layer">
334             /// <para>A layer.</para>
335             /// <para></para>
336             /// </param>
337             /// <exception cref="NotSupportedException">
```

```
            /// <para>Nested transactions not supported.</para>
            /// <para></para>
            /// </exception>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            public Transaction(UInt64LinksTransactionsLayer layer)
            {
                _layer = layer;
                if (_layer._currentTransactionId != 0)
                {
                    throw new NotSupportedException("Nested transactions not supported.");
                }
                IsCommitted = false;
                IsReverted = false;
                _transitions = new Queue<Transition>();
                SetCurrentTransaction(layer, this);
            }

            /// <summary>
            /// <para>
            /// Commits this instance.
            /// </para>
            /// <para></para>
            /// </summary>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            public void Commit()
            {
                EnsureTransactionAllowsWriteOperations(this);
                while (_transitions.Count > 0)
                {
                    var transition = _transitions.Dequeue();
                    _layer._transitions.Enqueue(transition);
                }
                _layer._lastCommitedTransactionId = _layer._currentTransactionId;
                IsCommitted = true;
            }

            /// <summary>
            /// <para>
            /// Reverts this instance.
            /// </para>
            /// <para></para>
            /// </summary>
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            private void Revert()
            {
                EnsureTransactionAllowsWriteOperations(this);
                var transitionsToRevert = new Transition[_transitions.Count];
                _transitions.CopyTo(transitionsToRevert, 0);
                for (var i = transitionsToRevert.Length - 1; i >= 0; i--)
                {
                    _layer.RevertTransition(transitionsToRevert[i]);
                }
                IsReverted = true;
            }

            /// <summary>
            /// <para>
            /// Sets the current transaction using the specified layer.
            /// </para>
            /// <para></para>
            /// </summary>
            /// <param name="layer">
            /// <para>The layer.</para>
            /// <para></para>
            /// </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>
        public static readonly TimeSpan DefaultPushDelay = TimeSpan.FromSeconds(0.1);

        /// <summary>
        /// <para>
        /// The log address.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly string _logAddress;
        /// <summary>
        /// <para>
        /// The log.
        /// </para>
        /// <para></para>
```

```
        /// </summary>
        private readonly FileStream _log;
        /// <summary>
        /// <para>
        /// The transitions.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly Queue<Transition> _transitions;
        /// <summary>
        /// <para>
        /// The unique timestamp factory.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly UniqueTimestampFactory _uniqueTimestampFactory;
        /// <summary>
        /// <para>
        /// The transitions pusher.
        /// </para>
        /// <para></para>
        /// </summary>
        private Task _transitionsPusher;
        /// <summary>
        /// <para>
        /// The last commited transition.
        /// </para>
        /// <para></para>
        /// </summary>
        private Transition _lastCommitedTransition;
        /// <summary>
        /// <para>
        /// The current transaction id.
        /// </para>
        /// <para></para>
        /// </summary>
        private ulong _currentTransactionId;
        /// <summary>
        /// <para>
        /// The current transaction transitions.
        /// </para>
        /// <para></para>
        /// </summary>
        private Queue<Transition> _currentTransactionTransitions;
        /// <summary>
        /// <para>
        /// The current transaction.
        /// </para>
        /// <para></para>
        /// </summary>
        private Transaction _currentTransaction;
        /// <summary>
        /// <para>
        /// The last commited transaction id.
        /// </para>
        /// <para></para>
        /// </summary>
        private ulong _lastCommitedTransactionId;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="UInt64LinksTransactionsLayer"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        /// <param name="logAddress">
        /// <para>A log address.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="ArgumentNullException">
        /// <para></para>
        /// <para></para>
        /// </exception>
        /// <exception cref="NotSupportedException">
```

```
572         /// <para>Database is damaged, autorecovery is not supported yet.</para>
573         /// <para></para>
574         /// </exception>
575         [MethodImpl(MethodImplOptions.AggressiveInlining)]
576         public UInt64LinksTransactionsLayer(ILinks<ulong> links, string logAddress)
577             : base(links)
578         {
579             if (string.IsNullOrWhiteSpace(logAddress))
580             {
581                 throw new ArgumentNullException(nameof(logAddress));
582             }
583             // В первой строке файла хранится последняя закоммиченную транзакцию.
584             // При запуске это используется для проверки удачного закрытия файла лога.
585             // In the first line of the file the last committed transaction is stored.
586             // On startup, this is used to check that the log file is successfully closed.
587             var lastCommitedTransition = FileHelpers.ReadFirstOrDefault<Transition>(logAddress);
588             var lastWrittenTransition = FileHelpers.ReadLastOrDefault<Transition>(logAddress);
589             if (!lastCommitedTransition.Equals(lastWrittenTransition))
590             {
591                 Dispose();
592                 throw new NotSupportedException("Database is damaged, autorecovery is not
                    ↪  supported yet.");
593             }
594             if (lastCommitedTransition == default)
595             {
596                 FileHelpers.WriteFirst(logAddress, lastCommitedTransition);
597             }
598             _lastCommitedTransition = lastCommitedTransition;
599             // TODO: Think about a better way to calculate or store this value
600             var allTransitions = FileHelpers.ReadAll<Transition>(logAddress);
601             _lastCommitedTransactionId = allTransitions.Length > 0 ? allTransitions.Max(x =>
                ↪  x.TransactionId) : 0;
602             _uniqueTimestampFactory = new UniqueTimestampFactory();
603             _logAddress = logAddress;
604             _log = FileHelpers.Append(logAddress);
605             _transitions = new Queue<Transition>();
606             _transitionsPusher = new Task(TransitionsPusher);
607             _transitionsPusher.Start();
608         }
609 
610         /// <summary>
611         /// <para>
612         /// Gets the link value using the specified link.
613         /// </para>
614         /// <para></para>
615         /// </summary>
616         /// <param name="link">
617         /// <para>The link.</para>
618         /// <para></para>
619         /// </param>
620         /// <returns>
621         /// <para>A list of ulong</para>
622         /// <para></para>
623         /// </returns>
624         [MethodImpl(MethodImplOptions.AggressiveInlining)]
625         public IList<ulong> GetLinkValue(ulong link) => _links.GetLink(link);
626 
627         /// <summary>
628         /// <para>
629         /// Creates the restrictions.
630         /// </para>
631         /// <para></para>
632         /// </summary>
633         /// <param name="restrictions">
634         /// <para>The restrictions.</para>
635         /// <para></para>
636         /// </param>
637         /// <returns>
638         /// <para>The created link index.</para>
639         /// <para></para>
640         /// </returns>
641         [MethodImpl(MethodImplOptions.AggressiveInlining)]
642         public override ulong Create(IList<ulong> restrictions)
643         {
644             var createdLinkIndex = _links.Create();
645             var createdLink = new Link<ulong>(_links.GetLink(createdLinkIndex));
646             CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
                ↪  default, createdLink));
```

```
647             return createdLinkIndex;
648         }
649 
650         /// <summary>
651         /// <para>
652         /// Updates the restrictions.
653         /// </para>
654         /// <para></para>
655         /// </summary>
656         /// <param name="restrictions">
657         /// <para>The restrictions.</para>
658         /// <para></para>
659         /// </param>
660         /// <param name="substitution">
661         /// <para>The substitution.</para>
662         /// <para></para>
663         /// </param>
664         /// <returns>
665         /// <para>The link index.</para>
666         /// <para></para>
667         /// </returns>
668         [MethodImpl(MethodImplOptions.AggressiveInlining)]
669         public override ulong Update(IList<ulong> restrictions, IList<ulong> substitution)
670         {
671             var linkIndex = restrictions[_constants.IndexPart];
672             var beforeLink = new Link<ulong>(_links.GetLink(linkIndex));
673             linkIndex = _links.Update(restrictions, substitution);
674             var afterLink = new Link<ulong>(_links.GetLink(linkIndex));
675             CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
                 ↪  beforeLink, afterLink));
676             return linkIndex;
677         }
678 
679         /// <summary>
680         /// <para>
681         /// Deletes the restrictions.
682         /// </para>
683         /// <para></para>
684         /// </summary>
685         /// <param name="restrictions">
686         /// <para>The restrictions.</para>
687         /// <para></para>
688         /// </param>
689         [MethodImpl(MethodImplOptions.AggressiveInlining)]
690         public override void Delete(IList<ulong> restrictions)
691         {
692             var link = restrictions[_constants.IndexPart];
693             var deletedLink = new Link<ulong>(_links.GetLink(link));
694             _links.Delete(link);
695             CommitTransition(new Transition(_uniqueTimestampFactory, _currentTransactionId,
                 ↪  deletedLink, default));
696         }
697 
698         /// <summary>
699         /// <para>
700         /// Gets the current transitions.
701         /// </para>
702         /// <para></para>
703         /// </summary>
704         /// <returns>
705         /// <para>A queue of transition</para>
706         /// <para></para>
707         /// </returns>
708         [MethodImpl(MethodImplOptions.AggressiveInlining)]
709         private Queue<Transition> GetCurrentTransitions() => _currentTransactionTransitions ??
             ↪  _transitions;
710 
711         /// <summary>
712         /// <para>
713         /// Commits the transition using the specified transition.
714         /// </para>
715         /// <para></para>
716         /// </summary>
717         /// <param name="transition">
718         /// <para>The transition.</para>
719         /// <para></para>
720         /// </param>
721         [MethodImpl(MethodImplOptions.AggressiveInlining)]
```

```
722         private void CommitTransition(Transition transition)
723         {
724             if (_currentTransaction != null)
725             {
726                 Transaction.EnsureTransactionAllowsWriteOperations(_currentTransaction);
727             }
728             var transitions = GetCurrentTransitions();
729             transitions.Enqueue(transition);
730         }
731 
732         /// <summary>
733         /// <para>
734         /// Reverts the transition using the specified transition.
735         /// </para>
736         /// <para></para>
737         /// </summary>
738         /// <param name="transition">
739         /// <para>The transition.</para>
740         /// <para></para>
741         /// </param>
742         [MethodImpl(MethodImplOptions.AggressiveInlining)]
743         private void RevertTransition(Transition transition)
744         {
745             if (transition.After.IsNull()) // Revert Deletion with Creation
746             {
747                 _links.Create();
748             }
749             else if (transition.Before.IsNull()) // Revert Creation with Deletion
750             {
751                 _links.Delete(transition.After.Index);
752             }
753             else // Revert Update
754             {
755                 _links.Update(new[] { transition.After.Index, transition.Before.Source,
                    ↪  transition.Before.Target });
756             }
757         }
758 
759         /// <summary>
760         /// <para>
761         /// Resets the current transation.
762         /// </para>
763         /// <para></para>
764         /// </summary>
765         [MethodImpl(MethodImplOptions.AggressiveInlining)]
766         private void ResetCurrentTransation()
767         {
768             _currentTransactionId = 0;
769             _currentTransactionTransitions = null;
770             _currentTransaction = null;
771         }
772 
773         /// <summary>
774         /// <para>
775         /// Pushes the transitions.
776         /// </para>
777         /// <para></para>
778         /// </summary>
779         [MethodImpl(MethodImplOptions.AggressiveInlining)]
780         private void PushTransitions()
781         {
782             if (_log == null || _transitions == null)
783             {
784                 return;
785             }
786             for (var i = 0; i < _transitions.Count; i++)
787             {
788                 var transition = _transitions.Dequeue();
789 
790                 _log.Write(transition);
791                 _lastCommitedTransition = transition;
792             }
793         }
794 
795         /// <summary>
796         /// <para>
797         /// Transitionses the pusher.
798         /// </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();
            }
```

```
877             base.Dispose(manual, wasDisposed);
878         }
879 
880         #endregion
881     }
882 }
```

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

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

```
67          /// <para></para>
68          /// </summary>
69          /// <typeparam name="TLink">
70          /// <para>The link.</para>
71          /// <para></para>
72          /// </typeparam>
73          /// <param name="action">
74          /// <para>The action.</para>
75          /// <para></para>
76          /// </param>
77          private static void Using<TLink>(Action<ILinks<TLink>> action)
78          {
79              using (var scope = new Scope<Types<HeapResizableDirectMemory,
                ↪  UnitedMemoryLinks<TLink>>>())
80              {
81                  action(scope.Use<ILinks<TLink>>());
82              }
83          }
84      }
85  }
```

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

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

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

```
1   using Xunit;
2
3   namespace Platform.Data.Doublets.Tests
4   {
5       /// <summary>
6       /// <para>
7       /// Represents the links constants tests.
8       /// </para>
9       /// <para></para>
10      /// </summary>
11      public static class LinksConstantsTests
12      {
13          /// <summary>
14          /// <para>
15          /// Tests that external references test.
16          /// </para>
17          /// <para></para>
18          /// </summary>
19          [Fact]
20          public static void ExternalReferencesTest()
21          {
22              LinksConstants<ulong> constants = new LinksConstants<ulong>((1, long.MaxValue),
                ↪  (long.MaxValue + 1UL, ulong.MaxValue));
```

```

            //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));
        }
    }
}
```

1.119 ./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;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the resizable direct memory links tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class ResizableDirectMemoryLinksTests
    {
        /// <summary>
        /// <para>
        /// The instance.
        /// </para>
        /// <para></para>
        /// </summary>
        private static readonly LinksConstants<ulong> _constants =
        ↪  Default<LinksConstants<ulong>>.Instance;

        /// <summary>
        /// <para>
        /// Tests that basic file mapped memory test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void BasicFileMappedMemoryTest()
        {
            var tempFilename = Path.GetTempFileName();
            using (var memoryAdapter = new UInt64UnitedMemoryLinks(tempFilename))
            {
                memoryAdapter.TestBasicMemoryOperations();
            }
            File.Delete(tempFilename);
        }

        /// <summary>
        /// <para>
        /// Tests that basic heap memory test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void BasicHeapMemoryTest()
        {
            using (var memory = new
            ↪  HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
            using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
            ↪  UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
            {
                memoryAdapter.TestBasicMemoryOperations();
            }
        }

        /// <summary>
        /// <para>
        /// Tests the basic memory operations using the specified memory adapter.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="memoryAdapter">
```

```
65          /// <para>The memory adapter.</para>
66          /// <para></para>
67          /// </param>
68          private static void TestBasicMemoryOperations(this ILinks<ulong> memoryAdapter)
69          {
70              var link = memoryAdapter.Create();
71              memoryAdapter.Delete(link);
72          }
73
74          /// <summary>
75          /// <para>
76          /// Tests that nonexistent references heap memory test.
77          /// </para>
78          /// <para></para>
79          /// </summary>
80          [Fact]
81          public static void NonexistentReferencesHeapMemoryTest()
82          {
83              using (var memory = new
                ↪ HeapResizableDirectMemory(UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
84              using (var memoryAdapter = new UInt64UnitedMemoryLinks(memory,
                ↪ UInt64UnitedMemoryLinks.DefaultLinksSizeStep))
85              {
86                  memoryAdapter.TestNonexistentReferences();
87              }
88          }
89
90          /// <summary>
91          /// <para>
92          /// Tests the nonexistent references using the specified memory adapter.
93          /// </para>
94          /// <para></para>
95          /// </summary>
96          /// <param name="memoryAdapter">
97          /// <para>The memory adapter.</para>
98          /// <para></para>
99          /// </param>
100         private static void TestNonexistentReferences(this ILinks<ulong> memoryAdapter)
101         {
102             var link = memoryAdapter.Create();
103             memoryAdapter.Update(link, ulong.MaxValue, ulong.MaxValue);
104             var resultLink = _constants.Null;
105             memoryAdapter.Each(foundLink =>
106             {
107                 resultLink = foundLink[_constants.IndexPart];
108                 return _constants.Break;
109             }, _constants.Any, ulong.MaxValue, ulong.MaxValue);
110             Assert.True(resultLink == link);
111             Assert.True(memoryAdapter.Count(ulong.MaxValue) == 0);
112             memoryAdapter.Delete(link);
113         }
114     }
115 }
```

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

```
1  using Xunit;
2  using Platform.Scopes;
3  using Platform.Memory;
4  using Platform.Data.Doublets.Decorators;
5  using Platform.Reflection;
6  using Platform.Data.Doublets.Memory.United.Generic;
7  using Platform.Data.Doublets.Memory.United.Specific;
8
9  namespace Platform.Data.Doublets.Tests
10 {
11     /// <summary>
12     /// <para>
13     /// Represents the scope tests.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class ScopeTests
18     {
19         /// <summary>
20         /// <para>
21         /// Tests that single dependency test.
22         /// </para>
23         /// <para></para>
24         /// </summary>
```

```
        [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);
            }
        }
    }
}
```

1.121 ./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>
```

```
14      /// </summary>
15      public unsafe static class SplitMemoryGenericLinksTests
16      {
17          /// <summary>
18          /// <para>
19          /// Tests that crud test.
20          /// </para>
21          /// <para></para>
22          /// </summary>
23          [Fact]
24          public static void CRUDTest()
25          {
26              Using<byte>(links => links.TestCRUDOperations());
27              Using<ushort>(links => links.TestCRUDOperations());
28              Using<uint>(links => links.TestCRUDOperations());
29              Using<ulong>(links => links.TestCRUDOperations());
30          }
31
32          /// <summary>
33          /// <para>
34          /// Tests that raw numbers crud test.
35          /// </para>
36          /// <para></para>
37          /// </summary>
38          [Fact]
39          public static void RawNumbersCRUDTest()
40          {
41              UsingWithExternalReferences<byte>(links => links.TestRawNumbersCRUDOperations());
42              UsingWithExternalReferences<ushort>(links => links.TestRawNumbersCRUDOperations());
43              UsingWithExternalReferences<uint>(links => links.TestRawNumbersCRUDOperations());
44              UsingWithExternalReferences<ulong>(links => links.TestRawNumbersCRUDOperations());
45          }
46
47          /// <summary>
48          /// <para>
49          /// Tests that multiple random creations and deletions test.
50          /// </para>
51          /// <para></para>
52          /// </summary>
53          [Fact]
54          public static void MultipleRandomCreationsAndDeletionsTest()
55          {
56              Using<byte>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
                ↪  MultipleRandomCreationsAndDeletions(16)); // Cannot use more because current
                ↪  implementation of tree cuts out 5 bits from the address space.
57              Using<ushort>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Te
                ↪  stMultipleRandomCreationsAndDeletions(100));
58              Using<uint>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Test
                ↪  MultipleRandomCreationsAndDeletions(100));
59              Using<ulong>(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().Tes
                ↪  tMultipleRandomCreationsAndDeletions(100));
60          }
61
62          /// <summary>
63          /// <para>
64          /// Usings the action.
65          /// </para>
66          /// <para></para>
67          /// </summary>
68          /// <typeparam name="TLink">
69          /// <para>The link.</para>
70          /// <para></para>
71          /// </typeparam>
72          /// <param name="action">
73          /// <para>The action.</para>
74          /// <para></para>
75          /// </param>
76          private static void Using<TLink>(Action<ILinks<TLink>> action)
77          {
78              using (var dataMemory = new HeapResizableDirectMemory())
79              using (var indexMemory = new HeapResizableDirectMemory())
80              using (var memory = new SplitMemoryLinks<TLink>(dataMemory, indexMemory))
81              {
82                  action(memory);
83              }
84          }
85
86          /// <summary>
```

```
        /// <para>
        /// Usings the with external references using the specified action.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="action">
        /// <para>The action.</para>
        /// <para></para>
        /// </param>
        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);
            }
        }
    }
}
```

1.122 ./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>
        /// Tests that multiple random creations and deletions test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void MultipleRandomCreationsAndDeletionsTest()
        {
            Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
            ↪  leRandomCreationsAndDeletions(500));
```

```
        }

        /// <summary>
        /// <para>
        /// Usings the action.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="action">
        /// <para>The action.</para>
        /// <para></para>
        /// </param>
        private static void Using(Action<ILinks<TLink>> action)
        {
            using (var dataMemory = new HeapResizableDirectMemory())
            using (var indexMemory = new HeapResizableDirectMemory())
            using (var memory = new UInt32SplitMemoryLinks(dataMemory, indexMemory))
            {
                action(memory);
            }
        }

        /// <summary>
        /// <para>
        /// Usings the with external references using the specified action.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="action">
        /// <para>The action.</para>
        /// <para></para>
        /// </param>
        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 UInt32SplitMemoryLinks(dataMemory, indexMemory,
            ↪  UInt32SplitMemoryLinks.DefaultLinksSizeStep, contants))
            {
                action(memory);
            }
        }
    }
}
```

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

```
using System;
using Xunit;
using Platform.Memory;
using Platform.Data.Doublets.Memory.Split.Specific;
using TLink = System.UInt64;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the split memory int 64 links tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public unsafe static class SplitMemoryUInt64LinksTests
    {
        /// <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>
```

```
33          /// <para></para>
34          /// </summary>
35          [Fact]
36          public static void RawNumbersCRUDTest()
37          {
38              UsingWithExternalReferences(links => links.TestRawNumbersCRUDOperations());
39          }
40
41          /// <summary>
42          /// <para>
43          /// Tests that multiple random creations and deletions test.
44          /// </para>
45          /// <para></para>
46          /// </summary>
47          [Fact]
48          public static void MultipleRandomCreationsAndDeletionsTest()
49          {
50              Using(links => links.DecorateWithAutomaticUniquenessAndUsagesResolution().TestMultip
                ↪ leRandomCreationsAndDeletions(500));
51          }
52
53          /// <summary>
54          /// <para>
55          /// Usings the action.
56          /// </para>
57          /// <para></para>
58          /// </summary>
59          /// <param name="action">
60          /// <para>The action.</para>
61          /// <para></para>
62          /// </param>
63          private static void Using(Action<ILinks<TLink>> action)
64          {
65              using (var dataMemory = new HeapResizableDirectMemory())
66              using (var indexMemory = new HeapResizableDirectMemory())
67              using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory))
68              {
69                  action(memory);
70              }
71          }
72
73          /// <summary>
74          /// <para>
75          /// Usings the with external references using the specified action.
76          /// </para>
77          /// <para></para>
78          /// </summary>
79          /// <param name="action">
80          /// <para>The action.</para>
81          /// <para></para>
82          /// </param>
83          private static void UsingWithExternalReferences(Action<ILinks<TLink>> action)
84          {
85              var contants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
86              using (var dataMemory = new HeapResizableDirectMemory())
87              using (var indexMemory = new HeapResizableDirectMemory())
88              using (var memory = new UInt64SplitMemoryLinks(dataMemory, indexMemory,
                ↪ UInt64SplitMemoryLinks.DefaultLinksSizeStep, contants))
89              {
90                  action(memory);
91              }
92          }
93      }
94  }
```

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

```
1   using System.Collections.Generic;
2   using Xunit;
3   using Platform.Ranges;
4   using Platform.Numbers;
5   using Platform.Random;
6   using Platform.Setters;
7   using Platform.Converters;
8
9   namespace Platform.Data.Doublets.Tests
10  {
11      /// <summary>
12      /// <para>
```

```
13      /// Represents the test extensions.
14      /// </para>
15      /// <para></para>
16      /// </summary>
17      public static class TestExtensions
18      {
19          /// <summary>
20          /// <para>
21          /// Tests the crud operations using the specified links.
22          /// </para>
23          /// <para></para>
24          /// </summary>
25          /// <typeparam name="T">
26          /// <para>The .</para>
27          /// <para></para>
28          /// </typeparam>
29          /// <param name="links">
30          /// <para>The links.</para>
31          /// <para></para>
32          /// </param>
33          public static void TestCRUDOperations<T>(this ILinks<T> links)
34          {
35              var constants = links.Constants;
36
37              var equalityComparer = EqualityComparer<T>.Default;
38
39              var zero = default(T);
40              var one = Arithmetic.Increment(zero);
41
42              // Create Link
43              Assert.True(equalityComparer.Equals(links.Count(), zero));
44
45              var setter = new Setter<T>(constants.Null);
46              links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
47
48              Assert.True(equalityComparer.Equals(setter.Result, constants.Null));
49
50              var linkAddress = links.Create();
51
52              var link = new Link<T>(links.GetLink(linkAddress));
53
54              Assert.True(link.Count == 3);
55              Assert.True(equalityComparer.Equals(link.Index, linkAddress));
56              Assert.True(equalityComparer.Equals(link.Source, constants.Null));
57              Assert.True(equalityComparer.Equals(link.Target, constants.Null));
58
59              Assert.True(equalityComparer.Equals(links.Count(), one));
60
61              // Get first link
62              setter = new Setter<T>(constants.Null);
63              links.Each(constants.Any, constants.Any, setter.SetAndReturnFalse);
64
65              Assert.True(equalityComparer.Equals(setter.Result, linkAddress));
66
67              // Update link to reference itself
68              links.Update(linkAddress, linkAddress, linkAddress);
69
70              link = new Link<T>(links.GetLink(linkAddress));
71
72              Assert.True(equalityComparer.Equals(link.Source, linkAddress));
73              Assert.True(equalityComparer.Equals(link.Target, linkAddress));
74
75              // Update link to reference null (prepare for delete)
76              var updated = links.Update(linkAddress, constants.Null, constants.Null);
77
78              Assert.True(equalityComparer.Equals(updated, linkAddress));
79
80              link = new Link<T>(links.GetLink(linkAddress));
81
82              Assert.True(equalityComparer.Equals(link.Source, constants.Null));
83              Assert.True(equalityComparer.Equals(link.Target, constants.Null));
84
85              // Delete link
86              links.Delete(linkAddress);
87
88              Assert.True(equalityComparer.Equals(links.Count(), zero));
89
90              setter = new Setter<T>(constants.Null);
91              links.Each(constants.Any, constants.Any, setter.SetAndReturnTrue);
92
```

```
            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>
        /// </param>
        /// <param name="maximumOperationsPerCycle">
        /// <para>The maximum operations per cycle.</para>
        /// <para></para>
        /// </param>
        public static void TestMultipleRandomCreationsAndDeletions<TLink>(this ILinks<TLink>
            links, int maximumOperationsPerCycle)
        {
            var comparer = Comparer<TLink>.Default;
            var addressToUInt64Converter = CheckedConverter<TLink, ulong>.Default;
            var uInt64ToAddressConverter = CheckedConverter<ulong, TLink>.Default;
            for (var N = 1; N < maximumOperationsPerCycle; N++)
            {
                var random = new System.Random(N);
                var created = 0UL;
                var deleted = 0UL;
                for (var i = 0; i < N; i++)
                {
                    var linksCount = addressToUInt64Converter.Convert(links.Count());
                    var createPoint = random.NextBoolean();
                    if (linksCount >= 2 && createPoint)
                    {
                        var linksAddressRange = new Range<ulong>(1, linksCount);
                        TLink source = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
                            ddressRange));
                        TLink target = uInt64ToAddressConverter.Convert(random.NextUInt64(linksA
                            ddressRange));
                            //-V3086
                        var resultLink = links.GetOrCreate(source, target);
                        if (comparer.Compare(resultLink,
                            uInt64ToAddressConverter.Convert(linksCount)) > 0)
                        {
                            created++;
                        }
                    }
                    else
                    {
                        links.Create();
                        created++;
                    }
                }
                Assert.True(created == addressToUInt64Converter.Convert(links.Count()));
                for (var i = 0; i < N; i++)
                {
                    TLink link = uInt64ToAddressConverter.Convert((ulong)i + 1UL);
                    if (links.Exists(link))
                    {
```

```
                        links.Delete(link);
                        deleted++;
                    }
                }
                Assert.True(addressToUInt64Converter.Convert(links.Count()) == 0L);
            }
        }
    }
}
```

1.125 ./csharp/Platform.Data.Doublets.Tests/Uint64LinksExtensionsTests.cs

```
using Platform.Data.Doublets.Memory;
using Platform.Data.Doublets.Memory.United.Generic;
using Platform.Data.Numbers.Raw;
using Platform.Memory;
using Platform.Numbers;
using Xunit;
using Xunit.Abstractions;
using TLink = System.UInt64;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// Represents the uint 64 links extensions tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class Uint64LinksExtensionsTests
    {
        /// <summary>
        /// <para>
        /// Creates the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A links of t link</para>
        /// <para></para>
        /// </returns>
        public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
        ↪  Platform.IO.TemporaryFile());

        /// <summary>
        /// <para>
        /// Creates the links using the specified data db filename.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="dataDBFilename">
        /// <para>The data db filename.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A links of t link</para>
        /// <para></para>
        /// </returns>
        public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
        {
            var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
            ↪  true);
            return new UnitedMemoryLinks<TLink>(new
            ↪  FileMappedResizableDirectMemory(dataDBFilename),
            ↪  UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
            ↪  IndexTreeType.Default);
        }
        /// <summary>
        /// <para>
        /// Tests that format structure with external reference test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void FormatStructureWithExternalReferenceTest()
        {
```

```
            ILinks<TLink> links = CreateLinks();
            TLink zero = default;
            var one = Arithmetic.Increment(zero);
            var markerIndex = one;
            var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
            var numberMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
            ↪  markerIndex));
            AddressToRawNumberConverter<TLink> addressToNumberConverter = new();
            var numberAddress = addressToNumberConverter.Convert(1);
            var numberLink = links.GetOrCreate(numberMarker, numberAddress);
            var linkNotation = links.FormatStructure(numberLink, link => link.IsFullPoint(),
            ↪  true);
            Assert.Equal("(3:(2:1 2) 18446744073709551615)", linkNotation);
        }
    }
}
```

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

```
using System;
using Xunit;
using Platform.Reflection;
using Platform.Memory;
using Platform.Scopes;
using Platform.Data.Doublets.Memory.United.Specific;
using TLink = System.UInt32;

namespace Platform.Data.Doublets.Tests
{
    /// <summary>
    /// <para>
    /// 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));
        }

        /// <summary>
        /// <para>
        /// Usings the action.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
        /// <param name="action">
        /// <para>The action.</para>
        /// <para></para>
        /// </param>
        private static void Using(Action<ILinks<TLink>> action)
        {
            using (var scope = new Scope<Types<HeapResizableDirectMemory,
            ↪  UInt32UnitedMemoryLinks>>())
            {
                action(scope.Use<ILinks<TLink>>());
            }
        }
    }
}
```

1.127 ./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]
        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));
        }

        /// <summary>
        /// <para>
        /// Usings the action.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="action">
        /// <para>The action.</para>
```

```
63          /// <para></para>
64          /// </param>
65          private static void Using(Action<ILinks<TLink>> action)
66          {
67              using (var scope = new Scope<Types<HeapResizableDirectMemory,
                ↪ UInt64UnitedMemoryLinks>>())
68              {
69                  action(scope.Use<ILinks<TLink>>());
70              }
71          }
72      }
73  }
```

Index