LinksPlatform's Platform.Communication.Protocol.Lino Class Library

1.1 ./csharp/Platform.Communication.Protocol.Lino/ILinksGroupListExtensions.cs

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

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

namespace Platform.Communication.Protocol.Lino
{
    /// <summary>
    /// <para>
    /// Represents the links group list extensions.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class ILinksGroupListExtensions
    {
        /// <summary>
        /// <para>
        /// Returns the links list using the specified groups.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="groups">
        /// <para>The groups.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The list.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static List<Link> ToLinksList(this IList<LinksGroup> groups)
        {
            var list = new List<Link>();
            for (var i = 0; i < groups.Count; i++)
            {
                groups[i].AppendToLinksList(list);
            }
            return list;
        }
    }
}
```

1.2 ./csharp/Platform.Communication.Protocol.Lino/IListExtensions.cs

```csharp
using Platform.Collections;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.CompilerServices;

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

namespace Platform.Communication.Protocol.Lino
{
    /// <summary>
    /// <para>
    /// Represents the list extensions.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class IListExtensions
    {
        /// <summary>
        /// <para>
        /// Formats the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string Format(this IList<Link> links) => string.Join(Environment.NewLine,
         ↪  links.Select(l => l.ToString()));

        /// <summary>
        /// <para>
        /// Formats the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>The links.</para>
        /// <para></para>
        /// </param>
        /// <param name="lessParentheses">
        /// <para>The less parentheses.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string Format(this IList<Link> links, bool lessParentheses)
        {
            if (lessParentheses == false)
            {
                return links.Format();
            }
            else
            {
                return string.Join(Environment.NewLine, links.Select(l =>
                 ↪  l.ToString().TrimSingle('(').TrimSingle(')')));
            }
        }
    }
}
```

1.3 ./csharp/Platform.Communication.Protocol.Lino/Link.cs

```
using System;
using System.Collections.Generic;
using System.Runtime.CompilerServices;
using System.Text;
using Platform.Collections;
using Platform.Collections.Lists;

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

namespace Platform.Communication.Protocol.Lino
{
    /// <summary>
    /// <para>
    /// The link.
    /// </para>
    /// <para></para>
    /// </summary>
    public struct Link : IEquatable<Link>
    {
        /// <summary>
        /// <para>
        /// The id.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly string Id;

        /// <summary>
        /// <para>
        /// The values.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly IList<Link> Values;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Link"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="id">
        /// <para>A id.</para>
        /// <para></para>
        /// </param>
        /// <param name="values">
        /// <para>A values.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(string id, IList<Link> values) => (Id, Values) = (id, values);

        /// <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<Link> values) : this(null, values) { }

        /// <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 Link[] values) : this(null, values) { }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="Link"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="id">
        /// <para>A id.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(string id) : this(id, null) { }

        /// <summary>
        /// <para>
        /// Returns the string.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override string ToString() => Values.IsNullOrEmpty() ? $"({Id})" :
         ↪  GetLinkValuesString();

        /// <summary>
        /// <para>
        /// Gets the link values string.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private string GetLinkValuesString() => Id == null ? $"({GetValuesString()})" :
         ↪  $"({Id}: {GetValuesString()})";

        /// <summary>
        /// <para>
        /// Gets the values string.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public string GetValuesString()
        {
            var sb = new StringBuilder();
            for (int i = 0; i < Values.Count; i++)
            {
                if (i > 0)
                {
                    sb.Append(' ');
                }
                sb.Append(GetValueString(Values[i]));
            }
            return sb.ToString();
        }

        /// <summary>
        /// <para>
        /// Simplifies this instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link Simplify()
        {
            if (Values.IsNullOrEmpty())
            {
                return this;
            }
            else if (Values.Count == 1)
            {
                return Values[0];
            }
            else
            {
                var newValues = new Link[Values.Count];
                for (int i = 0; i < Values.Count; i++)
                {
                    newValues[i] = Values[i].Simplify();
                }
                return new Link(Id, newValues);
            }
        }

        /// <summary>
        /// <para>
        /// Combines the other.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="other">
        /// <para>The other.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link Combine(Link other) => new Link(this, other);

        /// <summary>
        /// <para>
        /// Gets the value string using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string GetValueString(Link value) => value.ToLinkOrIdString();

        /// <summary>
        /// <para>
        /// Returns the link or id string.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public string ToLinkOrIdString() => Values.IsNullOrEmpty() ? Id : ToString();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator Link(string value) => new Link(value);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator Link((string, IList<Link>) value) => new
        ↪  Link(value.Item1, value.Item2);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator Link((Link, Link) value) => new Link(value.Item1,
        ↪  value.Item2);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator Link((Link, Link, Link) value) => new Link(value.Item1,
        ↪  value.Item2, value.Item3);

        /// <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 Link link ? Equals(link) : false;

        /// <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() => (Id, Values.GenerateHashCode()).GetHashCode();

        /// <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(Link other) => Id == other.Id && Values.EqualTo(other.Values);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static bool operator ==(Link left, Link right) => left.Equals(right);

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

1.4 ./csharp/Platform.Communication.Protocol.Lino/LinksGroup.cs

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

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

namespace Platform.Communication.Protocol.Lino
{
    /// <summary>
    /// <para>
    /// The links group.
    /// </para>
    /// <para></para>
    /// </summary>
    public struct LinksGroup : IEquatable<LinksGroup>
    {
        /// <summary>
        /// <para>
        /// Gets or sets the link value.
        /// </para>
        /// <para></para>
        /// </summary>
        public Link Link
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Gets or sets the groups value.
        /// </para>
        /// <para></para>
        /// </summary>
        public IList<LinksGroup> Groups
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksGroup"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>A link.</para>
        /// <para></para>
        /// </param>
        /// <param name="groups">
        /// <para>A groups.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksGroup(Link link, IList<LinksGroup> groups)
        {
            Link = link;
            Groups = groups;
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="LinksGroup"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>A link.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksGroup(Link link) : this(link, null) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator List<Link>(LinksGroup value) => value.ToLinksList();

        /// <summary>
        /// <para>
        /// Returns the links list.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The list.</para>
        /// <para></para>
        /// </returns>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public List<Link> ToLinksList()
        {
            var list = new List<Link>();
            AppendToLinksList(list);
            return list;
        }

        /// <summary>
        /// <para>
        /// Appends the to links list using the specified list.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="list">
        /// <para>The list.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void AppendToLinksList(List<Link> list) => AppendToLinksList(list, Link, this);

        /// <summary>
        /// <para>
        /// Appends the to links list using the specified list.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="list">
        /// <para>The list.</para>
        /// <para></para>
        /// </param>
        /// <param name="dependency">
        /// <para>The dependency.</para>
        /// <para></para>
        /// </param>
        /// <param name="group">
        /// <para>The group.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static void AppendToLinksList(List<Link> list, Link dependency, LinksGroup group)
        {
            list.Add(dependency);
            var groups = group.Groups;
            if (!groups.IsNullOrEmpty())
```

```
                {
                    for (int i = 0; i < groups.Count; i++)
                    {
                        var innerGroup = groups[i];
                        AppendToLinksList(list, dependency.Combine(innerGroup.Link), innerGroup);
                    }
                }
            }

            /// <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 LinksGroup linksGroup ?
            ↪  Equals(linksGroup) : false;

            /// <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() => (Link, Groups.GenerateHashCode()).GetHashCode();

            /// <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(LinksGroup other) => Link == other.Link &&
            ↪  Groups.EqualTo(other.Groups);

            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            public static bool operator ==(LinksGroup left, LinksGroup right) => left.Equals(right);

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

1.5 ./csharp/Platform.Communication.Protocol.Lino/_.cs

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

using System.Runtime.CompilerServices;

namespace Platform.Communication.Protocol.Lino
{
    /// <summary>
    /// <para>
    /// The .
    /// </para>
    /// <para></para>
    /// </summary>
    public struct _
    {
        /// <summary>
        /// <para>
        /// The link.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly Link Link;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="_"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="id">
        /// <para>A id.</para>
        /// <para></para>
        /// </param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public _(Link id) => Link = id;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator _(Link value) => new _(value);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator _(string id) => new _(id);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator _((string, Link) value) => new Link(value.Item1, new
        ↪  Link[] { value.Item2 });

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator _((string, Link, Link) value) => new Link(value.Item1,
        ↪  new Link[] { value.Item2, value.Item3 });

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator _((string, Link, Link, Link) value) => new
        ↪  Link(value.Item1, new Link[] { value.Item2, value.Item3, value.Item4 });

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static implicit operator Link(_ value) => value.Link;
    }
}
```

1.6 ./csharp/Platform.Communication.Protocol.Lino.Tests/ParserTests.cs

```
using Xunit;

namespace Platform.Communication.Protocol.Lino.Tests
{
    /// <summary>
    /// <para>
    /// Represents the parser tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public static class ParserTests
    {
        /// <summary>
        /// <para>
        /// Tests that single link test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void SingleLinkTest()
        {
            var source = @"(address: source target)";
            var parser = new Parser();
            var links = parser.Parse(source);
            var target = links.Format();
            Assert.Equal(source, target);
        }

        /// <summary>
        /// <para>
        /// Tests that two links test.
```

```csharp
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void TwoLinksTest()
        {
            var source = @"(first: x y)
(second: a b)";
            var parser = new Parser();
            var links = parser.Parse(source);
            var target = links.Format();
            Assert.Equal(source, target);
        }

        /// <summary>
        /// <para>
        /// Tests that parse and stringify test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void ParseAndStringifyTest()
        {
            var source = @"(papa (lovesMama: loves mama))
(son lovesMama)
(daughter lovesMama)
(all (love mama))";
            var parser = new Parser();
            var links = parser.Parse(source);
            var target = links.Format();
            Assert.Equal(source, target);
        }

        /// <summary>
        /// <para>
        /// Tests that bug test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void BugTest()
        {
            var source = @"(ignore conan-center-index repository)";
            var links = (new Platform.Communication.Protocol.Lino.Parser()).Parse(source);
            var target = links.Format();
            Assert.Equal(source,target);
        }

        /// <summary>
        /// <para>
        /// Tests that parse and stringify with less parentheses test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void ParseAndStringifyWithLessParenthesesTest()
        {
            var source = @"lovesMama: loves mama
papa lovesMama
son lovesMama
daughter lovesMama
all (love mama)";
            var parser = new Parser();
            var links = parser.Parse(source);
            var target = links.Format(lessParentheses: true);
            Assert.Equal(source, target);
        }

        /// <summary>
        /// <para>
        /// Tests that significant whitespace test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public static void SignificantWhitespaceTest()
        {
            var source = @"
users
    user1
        id
            43
        name
            first
                John
            last
                Williams
        location
            New York
        age
            23
    user2
        id
            56
        name
            first
                Igor
            middle
                Petrovich
            last
                Ivanov
        location
            Moscow
        age
            20";
            var target = @"(users)
(users user1)
((users user1) id)
(((users user1) id) 43)
((users user1) name)
(((users user1) name) first)
((((users user1) name) first) John)
(((users user1) name) last)
((((users user1) name) last) Williams)
((users user1) location)
(((users user1) location) (New York))
((users user1) age)
(((users user1) age) 23)
(users user2)
((users user2) id)
(((users user2) id) 56)
((users user2) name)
(((users user2) name) first)
((((users user2) name) first) Igor)
(((users user2) name) middle)
((((users user2) name) middle) Petrovich)
(((users user2) name) last)
((((users user2) name) last) Ivanov)
((users user2) location)
(((users user2) location) Moscow)
((users user2) age)
(((users user2) age) 20)";
            var parser = new Parser();
            var links = parser.Parse(source);
            var formattedLinks = links.Format();
            Assert.Equal(target, formattedLinks);
        }
    }
}
```

1.7 ./csharp/Platform.Communication.Protocol.Lino.Tests/TupleTests.cs

```
using System.Collections.Generic;
using Xunit;

namespace Platform.Communication.Protocol.Lino.Tests
{
    /// <summary>
    /// <para>
    /// Represents the tuple tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class TupleTests
    {
        /// <summary>
        /// <para>
        /// Tests that tuple to link test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void TupleToLinkTest()
        {
            var source = @"(papa (lovesMama: loves mama))
(son lovesMama)
(daughter lovesMama)
(all (love mama))";
            var parser = new Parser();
            var links = parser.Parse(source);
            var targetFromString = links.Format();

            IList<Link> constructedLinks = new List<Link>()
            {
                ("papa", (_)("lovesMama", "loves", "mama")),
                ("son", "lovesMama"),
                ("daughter", "lovesMama"),
                ("all", ("love", "mama")),
            };
            var targetFromTuples = constructedLinks.Format();
            Assert.Equal(targetFromString, targetFromTuples);
        }
    }
}
```

Index