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

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

```
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
{
    public static class ILinksGroupListExtensions
    {
        [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

```
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
{
    public static class IListExtensions
    {
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string Format(this IList<Link> links) => string.Join(Environment.NewLine,
         ↪  links.Select(l => l.ToString()));

        [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
{
    public struct Link : IEquatable<Link>
    {
        public string Id
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        public IList<Link> Values
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(string id, IList<Link> values) => (Id, Values) = (id, values);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(IList<Link> values) : this(null, values) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(params Link[] values) : this(null, values) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link(string id) : this(id, null) { }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override string ToString() => Values.IsNullOrEmpty() ? $"({Id})" :
        ↪   GetLinkValuesString();

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        private string GetLinkValuesString() => Id == null ? $"({GetValuesString()})" :
        ↪   $"({Id}: {GetValuesString()})";

        [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();
        }

        [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);
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public Link Combine(Link other) => new Link(this, other);

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public static string GetValueString(Link value) => value.ToLinkOrIdString();

        [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(ValueTuple<string, IList<Link>> value) => new
        ↪   Link(value.Item1, value.Item2);
        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override bool Equals(object obj) => obj is Link link ? Equals(link) : false;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override int GetHashCode() => (Id, Values.GenerateHashCode()).GetHashCode();

        [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
{
    public struct LinksGroup : IEquatable<LinksGroup>
    {
        public Link Link
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        public IList<LinksGroup> Groups
        {
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            get;
            [MethodImpl(MethodImplOptions.AggressiveInlining)]
            set;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksGroup(Link link, IList<LinksGroup> groups)
        {
            Link = link;
            Groups = groups;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public LinksGroup(Link link) : this(link, null) { }

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

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public List<Link> ToLinksList()
        {
            var list = new List<Link>();
            AppendToLinksList(list);
            return list;
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public void AppendToLinksList(List<Link> list) => AppendToLinksList(list, Link, this);

        [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);
                }
            }
        }

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override bool Equals(object obj) => obj is LinksGroup linksGroup ?
        ↪  Equals(linksGroup) : false;

        [MethodImpl(MethodImplOptions.AggressiveInlining)]
        public override int GetHashCode() => (Link, Groups.GenerateHashCode()).GetHashCode();

        [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.Tests/ParserTests.cs

```
using Xunit;

namespace Platform.Communication.Protocol.Lino.Tests
{
    public static class ParserTests
    {
        [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);
        }

        [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);
        }

        [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);
        }
    }
}
```

Index