

LinksPlatform's Platform.Data.Doublets.Lino Class Library

1.1 ./csharp/Platform.Data.Doublets.Lino/DefaultLinoStorage.cs

```
1 using System.Collections.Generic;
2 using Platform.Converters;
3 using LinoLink = Platform.Communication.Protocol.Lino.Link;
4
5 namespace Platform.Data.Doublets.Lino;
6
7 public class DefaultLinoStorage<TLinkAddress> : ILinoStorage<TLinkAddress>
8 {
9     private readonly ILinks<TLinkAddress> _storage;
10
11     public DefaultLinoStorage(ILinks<TLinkAddress> storage)
12     {
13         _storage = storage;
14     }
15
16     public void CreateLinks(IList<LinoLink> links)
17     {
18         var checkedConverter = CheckedConverter<ulong, TLinkAddress>.Default;
19         for (int i = 0; i < links.Count; i++)
20         {
21             _storage.Create();
22         }
23         for (int i = 0; i < links.Count; i++)
24         {
25             var index = ulong.Parse(links[i].Id);
26             var source = ulong.Parse(links[i].Values[0].Id);
27             var target = ulong.Parse(links[i].Values[1].Id);
28             _storage.Update(checkedConverter.Convert(index), checkedConverter.Convert(source),
29                 ↪ checkedConverter.Convert(target));
30         }
31
32     public IList<LinoLink> GetLinks()
33     {
34         var allLinks = _storage.All();
35         var linoLinks = new List<LinoLink>(allLinks.Count);
36         for (int i = 0; i < allLinks.Count; i++)
37         {
38             var link = new Link<TLinkAddress>(allLinks[i]);
39             var linoLink = new LinoLink(link.Index.ToString(), new List<LinoLink> {
40                 ↪ link.Source.ToString(), link.Target.ToString() });
41             linoLinks.Add(linoLink);
42         }
43         return linoLinks;
44     }
45 }
```

1.2 ./csharp/Platform.Data.Doublets.Lino/ILinoStorage.cs

```
1 using System.Collections.Generic;
2 using Platform.Communication.Protocol.Lino;
3
4 namespace Platform.Data.Doublets.Lino;
5
6 public interface ILinoStorage<TLinkAddress>
7 {
8     void CreateLinks(IList<Link> links);
9     IList<Link> GetLinks();
10 }
```

1.3 ./csharp/Platform.Data.Doublets.Lino/LinoDocumentsStorage.cs

```
1 using System;
2 using System.Collections.Generic;
3 using Platform.Collections.Stacks;
4 using Platform.Converters;
5 using Platform.Data.Doublets.CriterionMatchers;
6 using Platform.Data.Doublets.Numbers.Rational;
7 using Platform.Data.Doublets.Numbers.Raw;
8 using Platform.Data.Doublets.Sequences;
9 using Platform.Data.Doublets.Sequences.Converters;
10 using Platform.Data.Doublets.Sequences.HeightProviders;
11 using Platform.Data.Doublets.Sequences.Walkers;
12 using Platform.Data.Doublets.Unicode;
13 using Platform.Data.Numbers.Raw;
14 using Platform.Numbers;
15 using LinoLink = Platform.Communication.Protocol.Lino.Link;
16
17 namespace Platform.Data.Doublets.Lino;
```

```

18
19 public class LinoDocumentsStorage<TLinkAddress> : ILinoStorage<TLinkAddress> where TLinkAddress
    ↳ : struct
20 {
21     private readonly TLinkAddress _any;
22     private static readonly TLinkAddress Zero = default;
23     private static readonly TLinkAddress One = Arithmetic.Increment(Zero);
24
25     // public readonly IConverter<IList<TLinkAddress>?, TLinkAddress>
    ↳ ListToSequenceConverter;
26     private readonly TLinkAddress MeaningRoot;
27     private readonly EqualityComparer<TLinkAddress> _equalityComparer =
    ↳ EqualityComparer<TLinkAddress>.Default;
28     // Converters that are able to convert link's address (UInt64 value) to a raw number
    ↳ represented with another UInt64 value and back
29     public readonly RawNumberToAddressConverter<TLinkAddress> NumberToAddressConverter =
    ↳ new();
30     public readonly AddressToRawNumberConverter<TLinkAddress> AddressToNumberConverter =
    ↳ new();
31     // Converters between BigInteger and raw number sequence
32     public readonly BigIntegerToRawNumberSequenceConverter<TLinkAddress>
    ↳ BigIntegerToRawNumberSequenceConverter;
33     public readonly RawNumberSequenceToBigIntegerConverter<TLinkAddress>
    ↳ RawNumberSequenceToBigIntegerConverter;
34     // Converters between decimal and rational number sequence
35     public readonly DecimalToRationalConverter<TLinkAddress> DecimalToRationalConverter;
36     public readonly RationalToDecimalConverter<TLinkAddress> RationalToDecimalConverter;
37     public readonly DefaultSequenceRightHeightProvider<TLinkAddress>
    ↳ DefaultSequenceRightHeightProvider;
38     public readonly DefaultSequenceAppender<TLinkAddress> DefaultSequenceAppender;
39     public ILinks<TLinkAddress> Storage { get; }
40
41     public IConverter<string, TLinkAddress> StringToUnicodeSequenceConverter { get; }
42     public IConverter<TLinkAddress, string> UnicodeSequenceToStringConverter { get; }
43     public TLinkAddress DocumentMarker { get; }
44
45     public TLinkAddress ReferenceMarker { get; }
46     public TLinkAddress LinkMarker { get; }
47     public TLinkAddress LinkWithReferenceMarker { get; }
48     private TLinkAddress _markerIndex { get; set; }
49
50     private IConverter<IList<TLinkAddress>?, TLinkAddress> _listToSequenceConverter;
51
52     public LinoDocumentsStorage(ILinks<TLinkAddress> storage,
    ↳ IConverter<IList<TLinkAddress>?, TLinkAddress> listToSequenceConverter)
53     {
54         Storage = storage;
55         // ListToSequenceConverter = listToSequenceConverter;
56         // Initializes constants
57         _any = storage.Constants.Any;
58         var markerIndex = One;
59         MeaningRoot = storage.GetOrCreate(markerIndex, markerIndex);
60         var unicodeSymbolMarker = storage.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
    ↳ markerIndex));
61         var unicodeSequenceMarker = storage.GetOrCreate(MeaningRoot,
    ↳ Arithmetic.Increment(ref markerIndex));
62         DocumentMarker = storage.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
    ↳ markerIndex));
63         ReferenceMarker = storage.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
    ↳ markerIndex));
64         LinkMarker = storage.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
65         BalancedVariantConverter<TLinkAddress> balancedVariantConverter = new(storage);
66         TargetMatcher<TLinkAddress> unicodeSymbolCriterionMatcher = new(storage,
    ↳ unicodeSymbolMarker);
67         TargetMatcher<TLinkAddress> unicodeSequenceCriterionMatcher = new(storage,
    ↳ unicodeSequenceMarker);
68         CharToUnicodeSymbolConverter<TLinkAddress> charToUnicodeSymbolConverter =
69         new(storage, AddressToNumberConverter, unicodeSymbolMarker);
70         UnicodeSymbolToCharConverter<TLinkAddress> unicodeSymbolToCharConverter =
71         new(storage, NumberToAddressConverter, unicodeSymbolCriterionMatcher);
72         StringToUnicodeSequenceConverter = new CachingConverterDecorator<string,
    ↳ TLinkAddress>(
73             new StringToUnicodeSequenceConverter<TLinkAddress>(storage,
    ↳ charToUnicodeSymbolConverter,
74                 balancedVariantConverter, unicodeSequenceMarker));
75         RightSequenceWalker<TLinkAddress> sequenceWalker =
76         new(storage, new DefaultStack<TLinkAddress>(),
    ↳ unicodeSymbolCriterionMatcher.IsMatched);

```

```

77     UnicodeSequenceToStringConverter = new CachingConverterDecorator<TLinkAddress,
    ↪     string>(
78         new UnicodeSequenceToStringConverter<TLinkAddress>(storage,
    ↪         unicodeSequenceCriterionMatcher, sequenceWalker,
79             unicodeSymbolToCharConverter));
80     DecimalToRationalConverter = new(storage, BigIntegerToRawNumberSequenceConverter);
81     RationalToDecimalConverter = new(storage, RawNumberSequenceToBigIntegerConverter);
82     _listToSequenceConverter = listToSequenceConverter;
83 }
84
85 public TLinkAddress GetOrCreateReferenceLink(string content) =>
    ↪     Storage.GetOrCreate(ReferenceMarker,
    ↪     StringToUnicodeSequenceConverter.Convert(content));
86
87 public string ReadReference(TLinkAddress reference) =>
    ↪     ReadReference(Storage.GetLink(reference));
88
89 public string ReadReference(IList<TLinkAddress> reference)
90 {
91     var referenceLink = new Link<TLinkAddress>(reference);
92     if (!_equalityComparer.Equals(ReferenceMarker, referenceLink.Source))
93     {
94         throw new ArgumentException("The passed link is not a reference");
95     }
96     return UnicodeSequenceToStringConverter.Convert(referenceLink.Target);
97 }
98
99 public bool IsReference(TLinkAddress reference) =>
    ↪     IsReference(Storage.GetLink(reference));
100
101 public bool IsReference(IList<TLinkAddress> reference) =>
    ↪     _equalityComparer.Equals(ReferenceMarker, Storage.GetSource(reference));
102
103 public void CreateLinks(IList<LinoLink> links)
104 {
105     var sequenceList = new List<TLinkAddress>();
106     for (int i = 0; i < links.Count; i++)
107     {
108         sequenceList.Add(CreateLink(links[i]));
109     }
110     Storage.GetOrCreate(DocumentMarker, _listToSequenceConverter.Convert(sequenceList));
111 }
112
113 public TLinkAddress CreateLink(LinoLink link)
114 {
115     if (link.Id != null)
116     {
117         return CreateLinkWithReference(link);
118     }
119     var valuesSequence = CreateValuesSequence(link);
120     return Storage.GetOrCreate(LinkMarker, valuesSequence);
121 }
122
123 private TLinkAddress CreateLinkWithReference(LinoLink link)
124 {
125     TLinkAddress currentReference = GetOrCreateReferenceLink(link.Id);
126     if (link.Values == null)
127     {
128         return currentReference;
129     }
130     var valuesSequence = CreateValuesSequence(link);
131     var idWithValues = Storage.GetOrCreate(currentReference, valuesSequence);
132     return Storage.GetOrCreate(LinkWithReferenceMarker, idWithValues);
133 }
134
135 public TLinkAddress CreateValuesSequence(LinoLink parent)
136 {
137     var values = new List<TLinkAddress>(parent.Values.Count);
138     for (int i = 0; i < parent.Values.Count; i++)
139     {
140         var currentValue = parent.Values[i];
141         if (currentValue.Values != null)
142         {
143             var valueLink = CreateLink(currentValue);
144             values.Add(valueLink);
145             continue;
146         }
147         var currentValueReference = GetOrCreateReferenceLink(currentValue.Id);

```

```

148         values.Add(currentValueReference);
149     }
150     return _listToSequenceConverter.Convert(values);
151 }
152
153 public IList<LinoLink> GetLinks()
154 {
155     var resultLinks = new List<LinoLink>();
156     var any = Storage.Constants.Any;
157     bool IsElement(TLinkAddress linkIndex)
158     {
159         var source = Storage.GetSource(linkIndex);
160         return _equalityComparer.Equals(LinkMarker, source) |
            ↳ _equalityComparer.Equals(LinkWithReferenceMarker, source) ||
            ↳ Storage.IsPartialPoint(linkIndex);
161     }
162     var rightSequenceWalker = new RightSequenceWalker<TLinkAddress>(Storage, new
        ↳ DefaultStack<TLinkAddress>(), IsElement);
163     TLinkAddress linksSequence = default;
164     Storage.Each(DocumentMarker, any, link =>
165     {
166         linksSequence = Storage.GetTarget((IList<TLinkAddress>)link);
167         return Storage.Constants.Continue;
168     });
169     if (_equalityComparer.Equals(default, linksSequence))
170     {
171         throw new Exception("No one link in storage.");
172     }
173     var sequence = rightSequenceWalker.Walk(linksSequence);
174     foreach (var documentLink in sequence)
175     {
176         resultLinks.Add(GetLink(documentLink));
177     }
178     return resultLinks;
179 }
180
181 public LinoLink GetLink(TLinkAddress link) => GetLink(Storage.GetLink(link));
182
183 public LinoLink GetLink(IList<TLinkAddress> link)
184 {
185     string id = default;
186     var values = new List<LinoLink>();
187     var linkStruct = new Link<TLinkAddress>(link);
188     var isLink = _equalityComparer.Equals(LinkMarker, linkStruct.Source) ||
        ↳ _equalityComparer.Equals(LinkWithReferenceMarker, linkStruct.Source);
189     if (!isLink)
190     {
191         throw new Exception("The source of the passed link is not the link marker.");
192     }
193     bool IsElement(TLinkAddress linkIndex)
194     {
195         var source = Storage.GetSource(linkIndex);
196         return _equalityComparer.Equals(LinkMarker, source) ||
            ↳ _equalityComparer.Equals(LinkWithReferenceMarker, source) ||
            ↳ _equalityComparer.Equals(ReferenceMarker, source) ||
            ↳ Storage.IsPartialPoint(linkIndex);
197     }
198     var rightSequenceWalker = new RightSequenceWalker<TLinkAddress>(Storage, new
        ↳ DefaultStack<TLinkAddress>(), IsElement);
199     foreach (var currentLink in rightSequenceWalker.Walk(linkStruct.Target))
200     {
201         var currentLinkStruct = new Link<TLinkAddress>(Storage.GetLink(currentLink));
202         if (_equalityComparer.Equals(LinkMarker, currentLinkStruct.Source) ||
            ↳ _equalityComparer.Equals(LinkWithReferenceMarker, currentLinkStruct.Source))
203         {
204             var value = GetLink(currentLinkStruct);
205             values.Add(value);
206         }
207         else if (_equalityComparer.Equals(ReferenceMarker, currentLinkStruct.Source))
208         {
209             if (default == id)
210             {
211                 id = ReadReference(currentLinkStruct);
212             }
213             else
214             {
215                 var currentLinoLink = new LinoLink(ReadReference(currentLinkStruct));
216                 values.Add(currentLinoLink);

```

```

217     }
218 }
219
220 }
221     return new LinoLink(id, values);
222 }
223 }

```

1.4 ./csharp/Platform.Data.Doublets.Lino/LinoExporter.cs

```

1  using System.Collections.Generic;
2  using Platform.Communication.Protocol.Lino;
3
4  namespace Platform.Data.Doublets.Lino;
5
6  public class LinoExporter<TLinkAddress>
7  {
8      private readonly EqualityComparer<TLinkAddress> _equalityComparer =
9          ↪ EqualityComparer<TLinkAddress>.Default;
10     private readonly ILinoStorage<TLinkAddress> _linoDocumentsStorage;
11
12     public LinoExporter(ILinoStorage<TLinkAddress> linoDocumentsStorage)
13     {
14         _linoDocumentsStorage = linoDocumentsStorage;
15     }
16
17     public string GetAllLinks()
18     {
19         var allLinks = _linoDocumentsStorage.GetLinks();
20         return allLinks.Format();
21     }
22
23     // public void GetAllLinks(Stream outputStream)
24     // {
25     //     var allLinks = _linoDocumentsStorage.Storage.All();
26     //     var linksAsStrings = new Dictionary<TLinkAddress, string>(allLinks.Count);
27     //     for (int i = 0; i < allLinks.Count; i++)
28     //     {
29     //         var currentLink = allLinks[i];
30     //         linksAsStrings.Add(_linoDocumentsStorage.Storage.GetIndex(currentLink),
31     //             ↪ ReadLinkAsString(currentLink));
32     //     }
33     // }
34
35     // private LinoLink ReadLinkAsString(IList<TLinkAddress> link)
36     // {
37     //     string source = "";
38     //     string target = "";
39     //     var linkStruct = new Link<TLinkAddress>(link);
40     //     if (_linoDocumentsStorage.IsReference(linkStruct.Source))
41     //     {
42     //         source = _linoDocumentsStorage.ReadReference(linkStruct.Source);
43     //     }
44     //     if (_linoDocumentsStorage.IsReference(linkStruct.Target))
45     //     {
46     //         target = _linoDocumentsStorage.ReadReference(linkStruct.Target);
47     //     }
48     //     return new LinoLink(source, target);
49     // }

```

1.5 ./csharp/Platform.Data.Doublets.Lino/LinoExporterCli.cs

```

1  using System.IO;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Doublets.Sequences.Converters;
4  using Platform.IO;
5  using Platform.Memory;
6
7  namespace Platform.Data.Doublets.Lino;
8
9  public class LinoExporterCli<TLinkAddress> where TLinkAddress : struct
10 {
11     public void Run(params string[] args)
12     {
13         var argumentsIndex = 0;
14         var storageFilePath = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A path to a
15             ↪ links storage", args);
16         var notationFilePath = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A path to a
17             ↪ notation file", args);
18         var documentName = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A document name",
19             ↪ args);

```

```

17     using var linksMemory = new FileMappedResizableDirectMemory(storageFilePath);
18     var storage = new UnitedMemoryLinks<TLinkAddress>(linksMemory).DecorateWithAutomaticUniq
    ↪ uenessAndUsagesResolution();
19     ILinoStorage<TLinkAddress> linoStorage;
20     if (string.IsNullOrEmpty(documentName))
21     {
22         linoStorage = new DefaultLinoStorage<TLinkAddress>(storage);
23     }
24     else
25     {
26         linoStorage = new LinoDocumentsStorage<TLinkAddress>(storage, new
    ↪ BalancedVariantConverter<TLinkAddress>(storage));
27     }
28     var exporter = new LinoExporter<TLinkAddress>(linoStorage);
29     var allLinksNotation = exporter.GetAllLinks();
30     File.WriteAllText(notationTokenFilePath, allLinksNotation);
31 }
32 }

```

1.6 ./csharp/Platform.Data.Doublets.Lino/LinoImporter.cs

```

1  using Platform.Communication.Protocol.Lino;
2
3  namespace Platform.Data.Doublets.Lino;
4
5  public class LinoImporter<TLinkAddress>
6  {
7      private readonly ILinoStorage<TLinkAddress> _linoDocumentsStorage;
8      private readonly Parser _parser = new Parser();
9      public LinoImporter(ILinoStorage<TLinkAddress> linoDocumentsStorage)
10     {
11         _linoDocumentsStorage = linoDocumentsStorage;
12     }
13
14     public void Import(string content)
15     {
16         var linoLinks = _parser.Parse(content);
17         _linoDocumentsStorage.CreateLinks(linoLinks);
18     }
19
20     // public void Import(string content)
21     // {
22     //     var linoLinks = _parser.Parse(content);
23     //     for (int i = 0; i < linoLinks.Count; i++)
24     //     {
25     //         var linoLink = linoLinks[i];
26     //         Read(linoLink);
27     //     }
28     // }
29     //
30     // public TLinkAddress Read(Link parent)
31     // {
32     //     var left = parent.Values[0];
33     //     var right = parent.Values[1];
34     //     var source = left.Values != null ? Read(left) :
    ↪ _linoDocumentsStorage.GetOrCreateReferenceLink(left.Id);
35     //     var target = right.Values != null ? Read(right) :
    ↪ _linoDocumentsStorage.GetOrCreateReferenceLink(right.Id);
36     //     return _linoDocumentsStorage.Storage.GetOrCreate(source, target);
37     // }
38 }

```

1.7 ./csharp/Platform.Data.Doublets.Lino/LinoImporterCli.cs

```

1  using System.IO;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Doublets.Sequences.Converters;
4  using Platform.IO;
5  using Platform.Memory;
6
7  namespace Platform.Data.Doublets.Lino;
8
9  public class LinoImporterCli<TLinkAddress> where TLinkAddress : struct
10 {
11     public void Run(params string[] args)
12     {
13         var argumentsIndex = 0;
14         var notationFilePath = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A path to a
    ↪ notation file", args);
15         var storageFilePath = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A path to a
    ↪ links storage", args);

```

```

16     var documentName = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A document name",
17         ↪ args);
18     using var linksMemory = new FileMappedResizableDirectMemory(storageFilePath);
19     var storage = new UnitedMemoryLinks<TLinkAddress>(linksMemory).DecorateWithAutomaticUniq,
20         ↪ uenessAndUsagesResolution();
21     ILinoStorage<TLinkAddress> linoStorage;
22     if (string.IsNullOrEmpty(documentName))
23     {
24         linoStorage = new DefaultLinoStorage<TLinkAddress>(storage);
25     }
26     else
27     {
28         linoStorage = new LinoDocumentsStorage<TLinkAddress>(storage, new
29             ↪ BalancedVariantConverter<TLinkAddress>(storage));
30     }
31     var importer = new LinoImporter<TLinkAddress>(linoStorage);
32     var notation = File.ReadAllText(notationFilePath);
33     importer.Import(notation);
34 }
35 }

```

1.8 ./csharp/Platform.Data.Doublets.Lino.Tests/ImporterAndExporterCliTests.cs

```

1  using System;
2  using System.IO;
3  using Platform.IO;
4  using Xunit;
5  using TLinkAddress = System.UInt64;
6
7  namespace Platform.Data.Doublets.Lino.Tests;
8
9  public class ImporterAndExporterCliTests
10 {
11     [Theory]
12     [InlineData("(1: 1 1)")]
13     [InlineData("(1: 1 1)\n(2: 2 2)")]
14     [InlineData("(1: 2 2)")]
15     [InlineData("(1: 2 2)\n(2: 1 1)")]
16     public void DefaultLinoStorageTest(string notation)
17     {
18         var notationFilePath = TemporaryFiles.UseNew();
19         var linksStorageFilePath = TemporaryFiles.UseNew();
20         var exportedNotationFilePath = TemporaryFiles.UseNew();
21         File.WriteAllText(notationFilePath, notation);
22         new LinoImporterCli<TLinkAddress>().Run(notationFilePath, linksStorageFilePath, "");
23         new LinoExporterCli<TLinkAddress>().Run(linksStorageFilePath, exportedNotationFilePath,
24             ↪ "");
25         Assert.Equal(notation, File.ReadAllText(exportedNotationFilePath));
26     }
27
28     [Theory]
29     [InlineData("(1: 1 1)")]
30     [InlineData("(1: 1 1)\n(2: 2 2)")]
31     [InlineData("(2: 2 2)")]
32     [InlineData("(1: 2 2)")]
33     [InlineData("(1: 2 2)\n(2: 1 1)")]
34     [InlineData("(1: 2 (3: 3 3))")]
35     [InlineData("(1: 2 (3: 3 3))\n(2: 1 1)")]
36     [InlineData("(son: lovesMama)")]
37     [InlineData("(papa: (lovesMama: loves mama))")]
38     [InlineData("(papa: (lovesMama: loves mama) son lovesMama daughter lovesMama all (love:
39         ↪ mama))")]
40     public void LinoDocumentsStorageTest(string notation)
41     {
42         var notationFilePath = TemporaryFiles.UseNew();
43         var linksStorageFilePath = TemporaryFiles.UseNew();
44         var exportedNotationFilePath = TemporaryFiles.UseNew();
45         File.WriteAllText(notationFilePath, notation);
46         new LinoImporterCli<TLinkAddress>().Run(notationFilePath, linksStorageFilePath,
47             ↪ notationFilePath);
48         new LinoExporterCli<TLinkAddress>().Run(linksStorageFilePath, exportedNotationFilePath,
49             ↪ notationFilePath);
50         Assert.Equal(notation, File.ReadAllText(exportedNotationFilePath));
51     }
52 }

```

1.9 ./csharp/Platform.Data.Doublets.Lino.Tests/ImporterAndExporterTests.cs

```

1  using Platform.Converters;
2  using Platform.Data.Doublets.CriterionMatchers;

```

```

3 using Platform.Data.Doublets.Memory;
4 using Platform.Data.Doublets.Memory.United.Generic;
5 using Platform.Data.Doublets.Sequences.Converters;
6 using Platform.Data.Doublets.Unicode;
7 using Platform.Data.Numbers.Raw;
8 using Platform.Memory;
9 using Platform.Numbers;
10 using Xunit;
11 using TLinkAddress = System.UInt64;
12
13 namespace Platform.Data.Doublets.Lino.Tests;
14
15 public class ImporterAndExporterTests
16 {
17     public static ILinks<TLinkAddress> CreateLinks() => CreateLinks(new IO.TemporaryFile());
18
19     public static ILinks<TLinkAddress> CreateLinks(string dbFilename)
20     {
21         var linksConstants = new LinksConstants<TLinkAddress>(enableExternalReferencesSupport:
22             ↪ true);
23         return new UnitedMemoryLinks<TLinkAddress>(new HeapResizableDirectMemory(),
24             ↪ UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
25             ↪ IndexTreeType.Default);
26     }
27
28     // [InlineData("(1: 1 1)")]
29     // [InlineData("(1: 1 1)\n(2: 2 2)")]
30     // [InlineData("(1: 2 2)\n(2: 1 1)")]
31     // [Theory]
32     // public void Test1(string notation)
33     // {
34     //     var storage = CreateLinks();
35     //     TLinkAddress Zero = default;
36     //     TLinkAddress One = Arithmetic.Increment(Zero);
37     //     var markerIndex = One;
38     //     var meaningRoot = storage.GetOrCreate(markerIndex, markerIndex);
39     //     var unicodeSymbolMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
40     ↪ markerIndex));
41     //     var unicodeSequenceMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
42     ↪ markerIndex));
43     //     var referenceMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
44     ↪ markerIndex));
45     //     TargetMatcher<TLinkAddress> unicodeSymbolCriterionMatcher = new(storage,
46     ↪ unicodeSymbolMarker);
47     //     TargetMatcher<TLinkAddress> unicodeSequenceCriterionMatcher = new(storage,
48     ↪ unicodeSequenceMarker);
49     //     AddressToRawNumberConverter<TLinkAddress> addressToNumberConverter = new();
50     //     RawNumberToAddressConverter<TLinkAddress> numberToAddressConverter = new();
51     //     CharToUnicodeSymbolConverter<TLinkAddress> charToUnicodeSymbolConverter =
52     ↪ new(storage, addressToNumberConverter, unicodeSymbolMarker);
53     //     UnicodeSymbolToCharConverter<TLinkAddress> unicodeSymbolToCharConverter =
54     ↪ new(storage, numberToAddressConverter, unicodeSymbolCriterionMatcher);
55     //     var balancedVariantConverter = new BalancedVariantConverter<TLinkAddress>(storage);
56     //     var stringToUnicodeSequenceConverter = new CachingConverterDecorator<string,
57     ↪ TLinkAddress>(new StringToUnicodeSequenceConverter<TLinkAddress>(storage,
58     ↪ charToUnicodeSymbolConverter, balancedVariantConverter, unicodeSequenceMarker));
59     //     ILinoDocumentsStorage<TLinkAddress> linoDocumentsStorage = new
60     ↪ LinoDocumentsStorage<TLinkAddress>(storage);
61     //     LinoImporter<TLinkAddress> linoImporter = new(linoDocumentsStorage);
62     //     linoImporter.Import(notation);
63     //     var linoExporter = new LinoExporter<TLinkAddress>(linoDocumentsStorage);
64     // }
65
66     [InlineData("(1: 1 1)")]
67     [InlineData("(1: 1 1)\n(2: 2 2)")]
68     [InlineData("(1: 2 2)")]
69     [InlineData("(1: 2 2)\n(2: 1 1)")]
70     [Theory]
71     public void LinoStorageTest(string notation)
72     {
73         var storage = CreateLinks();
74         var linoStorage = new DefaultLinoStorage<TLinkAddress>(storage);
75         var importer = new LinoImporter<TLinkAddress>(linoStorage);
76         importer.Import(notation);
77         var exporter = new LinoExporter<TLinkAddress>(linoStorage);
78         var exportedLinks = exporter.GetAllLinks();
79         Assert.Equal(notation, exportedLinks);
80     }
81 }

```



```

70
71 [InlineData("(1: 1 1)")]
72 [InlineData("(1: 1 1)\n(2: 2 2)")]
73 [InlineData("(2: 2 2)")]
74 [InlineData("(1: 2 2)")]
75 [InlineData("(1: 2 2)\n(2: 1 1)")]
76 [InlineData("(1: 2 (3: 3 3))")]
77 [InlineData("(1: 2 (3: 3 3))\n(2: 1 1)")]
78 [InlineData("(son: lovesMama)")]
79 [InlineData("(papa: (lovesMama: loves mama))")]
80 [InlineData("(papa: (lovesMama: loves mama) son lovesMama daughter lovesMama all (love:
↪ mama))")]
81 [Theory]
82 public void LinoDocumentStorageTest(string notation)
83 {
84     var storage = CreateLinks();
85     var linoStorage = new LinoDocumentsStorage<TLinkAddress>(storage, new
↪     BalancedVariantConverter<ulong>(storage));
86     var importer = new LinoImporter<TLinkAddress>(linoStorage);
87     importer.Import(notation);
88     var exporter = new LinoExporter<TLinkAddress>(linoStorage);
89     var exportedLinks = exporter.GetAllLinks();
90     Assert.Equal(notation, exportedLinks);
91 }
92
93 // public void CreateLinksTest()
94 // {
95 //     var storage = CreateLinks();
96 //     var linoStorage = new DefaultLinoStorage<TLinkAddress>(storage);
97 //     linoStorage.CreateLinks();
98 // }
99 //
100 // public void GetLinksTest()
101 // {
102 //     var storage = CreateLinks();
103 //     var linoStorage = new DefaultLinoStorage<TLinkAddress>(storage);
104 //     linoStorage.GetLinks();
105 // }
106 }

```

Index

./csharp/Platform.Data.Doublets.Lino.Tests/ImporterAndExporterCliTests.cs, 7
./csharp/Platform.Data.Doublets.Lino.Tests/ImporterAndExporterTests.cs, 7
./csharp/Platform.Data.Doublets.Lino/DefaultLinoStorage.cs, 1
./csharp/Platform.Data.Doublets.Lino/ILinoStorage.cs, 1
./csharp/Platform.Data.Doublets.Lino/LinoDocumentsStorage.cs, 1
./csharp/Platform.Data.Doublets.Lino/LinoExporter.cs, 5
./csharp/Platform.Data.Doublets.Lino/LinoExporterCli.cs, 5
./csharp/Platform.Data.Doublets.Lino/LinoImporter.cs, 6
./csharp/Platform.Data.Doublets.Lino/LinoImporterCli.cs, 6