

## 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
33     public IList<LinoLink> GetLinks()
34     {
35         var allLinks = _storage.All();
36         var linoLinks = new List<LinoLink>(allLinks.Count);
37         for (int i = 0; i < allLinks.Count; i++)
38         {
39             var link = new Link<TLinkAddress>(allLinks[i]);
40             var linoLink = new LinoLink(link.Index.ToString(), new List<LinoLink> {
41                 ↪ link.Source.ToString(), link.Target.ToString() });
42             linoLinks.Add(linoLink);
43         }
44         return linoLinks;
45     }
46 }
```

### 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 System.Linq;
4 using Platform.Collections.Stacks;
5 using Platform.Converters;
6 using Platform.Data.Doublets.CriterionMatchers;
7 using Platform.Data.Doublets.Numbers.Rational;
8 using Platform.Data.Doublets.Numbers.Raw;
9 using Platform.Data.Doublets.Sequences.Converters;
10 using Platform.Data.Doublets.Sequences.Walkers;
11 using Platform.Data.Doublets.Unicode;
12 using Platform.Data.Numbers.Raw;
13 using Platform.Numbers;
14 using LinoLink = Platform.Communication.Protocol.Lino.Link;
15
16 namespace Platform.Data.Doublets.Lino;
17
18 public class LinoDocumentsStorage<TLinkAddress> : ILinoStorage<TLinkAddress>
```

```

19 {
20     private static readonly TLinkAddress Zero = default!;
21     private static readonly TLinkAddress One = Arithmetic.Increment(Zero);
22
23     private readonly EqualityComparer<TLinkAddress> _equalityComparer =
24         ↳ EqualityComparer<TLinkAddress>.Default;
25
26     // Converters that are able to convert link's address (UInt64 value) to a raw number
27     ↳ represented with another UInt64 value and back
28     private readonly RawNumberToAddressConverter<TLinkAddress> _numberToAddressConverter = new();
29     private readonly AddressToRawNumberConverter<TLinkAddress> _addressToNumberConverter = new();
30
31     private ILinks<TLinkAddress> Storage { get; }
32
33     private IConverter<string, TLinkAddress> StringToUnicodeSequenceConverter { get; }
34
35     private IConverter<TLinkAddress, string> UnicodeSequenceToStringConverter { get; }
36
37     public TLinkAddress DocumentMarker { get; }
38
39     public TLinkAddress ReferenceMarker { get; }
40
41     public TLinkAddress LinkWithoutIdMarker { get; }
42
43     public TLinkAddress LinkWithIdMarker { get; }
44
45     private readonly IConverter<IList<TLinkAddress>?, TLinkAddress> _listToSequenceConverter;
46
47     public LinoDocumentsStorage(ILinks<TLinkAddress> storage, IConverter<IList<TLinkAddress>?,
48         ↳ TLinkAddress> listToSequenceConverter)
49     {
50         Storage = storage;
51         // Initializes constants
52         var markerIndex = One;
53         var meaningRoot = storage.GetOrCreate(markerIndex, markerIndex);
54         var unicodeSymbolMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
55             ↳ markerIndex));
56         var unicodeSequenceMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
57             ↳ markerIndex));
58         DocumentMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref markerIndex));
59         ReferenceMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
60             ↳ markerIndex));
61         LinkWithoutIdMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
62             ↳ markerIndex));
63         LinkWithIdMarker = storage.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
64             ↳ markerIndex));
65         BalancedVariantConverter<TLinkAddress> balancedVariantConverter = new(storage);
66         TargetMatcher<TLinkAddress> unicodeSymbolCriterionMatcher = new(storage,
67             ↳ unicodeSymbolMarker);
68         TargetMatcher<TLinkAddress> unicodeSequenceCriterionMatcher = new(storage,
69             ↳ unicodeSequenceMarker);
70         CharToUnicodeSymbolConverter<TLinkAddress> charToUnicodeSymbolConverter = new(storage,
71             ↳ _addressToNumberConverter, unicodeSymbolMarker);
72         UnicodeSymbolToCharConverter<TLinkAddress> unicodeSymbolToCharConverter = new(storage,
73             ↳ _numberToAddressConverter, unicodeSymbolCriterionMatcher);
74         StringToUnicodeSequenceConverter = new CachingConverterDecorator<string,
75             ↳ TLinkAddress>(new StringToUnicodeSequenceConverter<TLinkAddress>(storage,
76                 ↳ charToUnicodeSymbolConverter, balancedVariantConverter, unicodeSequenceMarker));
77         RightSequenceWalker<TLinkAddress> sequenceWalker = new(storage, new
78             ↳ DefaultStack<TLinkAddress>(), unicodeSymbolCriterionMatcher.IsMatched);
79         UnicodeSequenceToStringConverter = new CachingConverterDecorator<TLinkAddress,
80             ↳ string>(new UnicodeSequenceToStringConverter<TLinkAddress>(storage,
81                 ↳ unicodeSequenceCriterionMatcher, sequenceWalker, unicodeSymbolToCharConverter));
82         _listToSequenceConverter = listToSequenceConverter;
83     }
84
85     private bool IsLinkWithId(TLinkAddress linkAddress) =>
86         ↳ IsLinkWithId(Storage.GetLink(linkAddress));
87
88     private bool IsLinkWithoutId(TLinkAddress linkAddress) =>
89         ↳ IsLinkWithoutId(Storage.GetLink(linkAddress));
90
91     private bool IsLink(TLinkAddress linkAddress)
92     {
93         var link = Storage.GetLink(linkAddress);
94         return IsLinkWithId(link) || IsLinkWithoutId(link);
95     }
96
97 }

```

```

78     private bool IsLinkWithId(IList<TLinkAddress> link)
79     {
80         var source = Storage.GetSource(link);
81         return _equalityComparer.Equals(LinkWithIdMarker, source);
82     }
83
84     private bool IsLinkWithoutId(IList<TLinkAddress> link)
85     {
86         var source = Storage.GetSource(link);
87         return _equalityComparer.Equals(LinkWithoutIdMarker, source);
88     }
89
90     private bool IsLink(IList<TLinkAddress> link) => IsLinkWithId(link) || IsLinkWithoutId(link);
91
92     private bool IsReference(TLinkAddress reference)
93     {
94         var link = Storage.GetLink(reference);
95         return IsReference(link);
96     }
97
98     private bool IsReference(IList<TLinkAddress> reference)
99     {
100         var source = Storage.GetSource(reference);
101         return _equalityComparer.Equals(ReferenceMarker, source);
102     }
103
104
105     private TLinkAddress GetOrCreateReference(string content)
106     {
107         var sequence = StringToUnicodeSequenceConverter.Convert(content);
108         return Storage.GetOrCreate(ReferenceMarker, sequence);
109     }
110
111     private string ReadReference(TLinkAddress reference) =>
112     ↪ ReadReference(Storage.GetLink(reference));
113
114     private string ReadReference(IList<TLinkAddress> reference)
115     {
116         var referenceLink = new Link<TLinkAddress>(reference);
117         if (!_equalityComparer.Equals(ReferenceMarker, referenceLink.Source))
118         {
119             throw new ArgumentException("The passed link is not a reference");
120         }
121         return UnicodeSequenceToStringConverter.Convert(referenceLink.Target);
122     }
123
124     public void CreateLinks(IList<LinoLink> links)
125     {
126         var sequenceList = new List<TLinkAddress>();
127         for (int i = 0; i < links.Count; i++)
128         {
129             sequenceList.Add(CreateLink(links[i]));
130         }
131         Storage.GetOrCreate(DocumentMarker, _listToSequenceConverter.Convert(sequenceList));
132     }
133
134     private TLinkAddress CreateLink(LinoLink link)
135     {
136         if (link.Id != null)
137         {
138             return CreateLinkWithId(link);
139         }
140         var valuesSequence = CreateValuesSequence(link);
141         return Storage.GetOrCreate(LinkWithoutIdMarker, valuesSequence);
142     }
143
144     private TLinkAddress CreateLinkWithId(LinoLink link)
145     {
146         var currentReference = GetOrCreateReference(link.Id);
147         if (link.Values == null)
148         {
149             return Storage.GetOrCreate(LinkWithIdMarker, currentReference);
150         }
151         var valuesSequence = CreateValuesSequence(link);
152         var idWithValues = Storage.GetOrCreate(currentReference, valuesSequence);
153         return Storage.GetOrCreate(LinkWithIdMarker, idWithValues);
154     }
155
156     private TLinkAddress CreateValuesSequence(LinoLink parent)

```

```

156 {
157     var values = new List<TLinkAddress>(parent.Values.Count);
158     for (int i = 0; i < parent.Values.Count; i++)
159     {
160         var currentValue = parent.Values[i];
161         if (currentValue.Values != null)
162         {
163             var valueLink = CreateLink(currentValue);
164             values.Add(valueLink);
165             continue;
166         }
167         var currentValueReference = GetOrCreateReference(currentValue.Id);
168         values.Add(currentValueReference);
169     }
170     return _listToSequenceConverter.Convert(values);
171 }
172
173 private TLinkAddress GetDocumentSequence()
174 {
175     var constants = Storage.Constants;
176     var any = constants.Any;
177     TLinkAddress documentLinksSequence = default;
178     Storage.Each(new Link<TLinkAddress>(any, DocumentMarker, any), link =>
179     {
180         documentLinksSequence = Storage.GetTarget((IList<TLinkAddress>)link);
181         return constants.Continue;
182     });
183     if (_equalityComparer.Equals(default, documentLinksSequence))
184     {
185         throw new Exception("No document links in the storage.");
186     }
187     return documentLinksSequence;
188 }
189
190 public IList<LinoLink> GetLinks()
191 {
192     var resultLinks = new List<LinoLink>();
193     bool IsElement(TLinkAddress linkIndex)
194     {
195         var source = Storage.GetSource(linkIndex);
196         return _equalityComparer.Equals(LinkWithoutIdMarker, source) |
197             ↳ _equalityComparer.Equals(LinkWithIdMarker, source) ||
198             ↳ Storage.IsPartialPoint(linkIndex);
199     }
200     var rightSequenceWalker = new RightSequenceWalker<TLinkAddress>(Storage, new
201     ↳ DefaultStack<TLinkAddress>(), IsElement);
202     var documentLinksSequence = GetDocumentSequence();
203     var documentLinks = rightSequenceWalker.Walk(documentLinksSequence);
204     foreach (var documentLink in documentLinks)
205     {
206         resultLinks.Add(GetLink(documentLink));
207     }
208     return resultLinks;
209 }
210
211 private bool IsLinkOrReferenceOrPartialPoint(TLinkAddress linkIndex)
212 {
213     var link = Storage.GetLink(linkIndex);
214     return IsLink(link) || IsReference(link) || Storage.IsPartialPoint(linkIndex);
215 }
216
217 private LinoLink GetLinkWithId(IList<TLinkAddress> linkWithId)
218 {
219     string id;
220     var values = new List<LinoLink>();
221     var linkStruct = new Link<TLinkAddress>(linkWithId);
222     if (IsReference(linkStruct.Target))
223     {
224         id = ReadReference(linkStruct.Target);
225         return new LinoLink(id);
226     }
227     var rightSequenceWalker = new RightSequenceWalker<TLinkAddress>(Storage, new
228     ↳ DefaultStack<TLinkAddress>(), IsLinkOrReferenceOrPartialPoint);
229     using var enumerator = rightSequenceWalker.Walk(linkStruct.Target).GetEnumerator();
230     enumerator.MoveNext();
231     id = ReadReference(enumerator.Current);
232     while (enumerator.MoveNext())
233     {

```

```

230     var currentValueStruct = new Link<TLinkAddress>(Storage.GetLink(enumerator.Current));
231     if (IsLink(currentValueStruct))
232     {
233         var value = GetLink(currentValueStruct);
234         values.Add(value);
235     }
236     else if (IsReference(currentValueStruct))
237     {
238         var reference = ReadReference(currentValueStruct);
239         var currentLinoLink = new LinoLink(reference);
240         values.Add(currentLinoLink);
241     }
242 }
243 return new LinoLink(id, values);
244 }
245
246 public LinoLink GetLinkWithoutId(IList<TLinkAddress> linkWithoutId)
247 {
248     var values = new List<LinoLink>();
249     var linkStruct = new Link<TLinkAddress>(linkWithoutId);
250     var rightSequenceWalker = new RightSequenceWalker<TLinkAddress>(Storage, new
    ↪ DefaultStack<TLinkAddress>(), IsLinkOrReferenceOrPartialPoint);
251     foreach (var currentValue in rightSequenceWalker.Walk(linkStruct.Target))
252     {
253         var currentValueStruct = new Link<TLinkAddress>(Storage.GetLink(currentValue));
254         if (IsLink(currentValue))
255         {
256             var value = GetLink(currentValueStruct);
257             values.Add(value);
258         }
259         else if (IsReference(currentValueStruct))
260         {
261             var currentLinoLink = new LinoLink(ReadReference(currentValueStruct));
262             values.Add(currentLinoLink);
263         }
264     }
265     return new LinoLink(null, values);
266 }
267
268 public LinoLink GetLink(TLinkAddress link) => GetLink(Storage.GetLink(link));
269
270 public LinoLink GetLink(IList<TLinkAddress> link)
271 {
272     var linkStruct = new Link<TLinkAddress>(link);
273     if (IsLinkWithId(linkStruct))
274     {
275         return GetLinkWithId(linkStruct);
276     }
277     if (IsLinkWithoutId(linkStruct))
278     {
279         return GetLinkWithoutId(linkStruct);
280     }
281     throw new Exception("The passed argument is not a link");
282 }
283 }

```

#### 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 =
    ↪ EqualityComparer<TLinkAddress>.Default;
9     private readonly ILinoStorage<TLinkAddress> _linoDocumentsStorage;
10
11     public LinoExporter(ILinoStorage<TLinkAddress> linoDocumentsStorage)
12     {
13         _linoDocumentsStorage = linoDocumentsStorage;
14     }
15
16     public string GetAllLinks()
17     {
18         var allLinks = _linoDocumentsStorage.GetLinks();
19         return allLinks.Format();
20     }
21 }

```

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

```
1 using System;
2 using System.IO;
3 using Platform.Data.Doublets.Memory.United.Generic;
4 using Platform.Data.Doublets.Sequences.Converters;
5 using Platform.IO;
6 using Platform.Memory;
7
8 namespace Platform.Data.Doublets.Lino;
9
10 public class LinoExporterCli<TLinkAddress> where TLinkAddress : struct
11 {
12     public void Run(params string[] args)
13     {
14         var argumentsIndex = 0;
15         var storageFilePath = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A path to a
16             ↪ links storage", args);
17         var notationFilePath = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A path to a
18             ↪ notation file", args);
19         var documentName = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A document name",
20             ↪ args);
21         using var linksMemory = new FileMappedResizableDirectMemory(storageFilePath);
22         var storage = new UnitedMemoryLinks<TLinkAddress>(linksMemory).DecorateWithAutomaticUniq
23             ↪ uenessAndUsagesResolution();
24         ILinoStorage<TLinkAddress> linoStorage;
25         if (String.IsNullOrEmpty(documentName))
26         {
27             linoStorage = new DefaultLinoStorage<TLinkAddress>(storage);
28         }
29         else
30         {
31             linoStorage = new LinoDocumentsStorage<TLinkAddress>(storage, new
32                 ↪ BalancedVariantConverter<TLinkAddress>(storage));
33         }
34         var exporter = new LinoExporter<TLinkAddress>(linoStorage);
35         var allLinksNotation = exporter.GetAllLinks();
36         File.WriteAllText(notationFilePath, allLinksNotation);
37     }
38 }
```

### 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
10     public LinoImporter(ILinoStorage<TLinkAddress> linoDocumentsStorage)
11     {
12         _linoDocumentsStorage = linoDocumentsStorage;
13     }
14
15     public void Import(string content)
16     {
17         var linoLinks = _parser.Parse(content);
18         _linoDocumentsStorage.CreateLinks(linoLinks);
19     }
20
21 }
```

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

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

```

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

```

## 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         Assert.Equal(notation, File.ReadAllText(exportedNotationFilePath));
25     }
26
27     [Theory]
28     [InlineData("(1: 1 1)")]
29     [InlineData("(1: 1 1)\n(2: 2 2)")]
30     [InlineData("(2: 2 2)")]
31     [InlineData("(1: 2 2)")]
32     [InlineData("(1: 2 2)\n(2: 1 1)")]
33     [InlineData("(1: 2 (3: 3 3))")]
34     [InlineData("(1: 2 (3: 3 3))\n(2: 1 1)")]
35     [InlineData("(son: lovesMama)")]
36     [InlineData("(papa: (lovesMama: loves mama))")]
37     [InlineData("@(papa (lovesMama: loves mama))
38 (son lovesMama)
39 (daughter lovesMama)
40 (all (love mama))")]
41     public void LinoDocumentsStorageTest(string notation)
42     {
43         var notationFilePath = TemporaryFiles.UseNew();
44         var linksStorageFilePath = TemporaryFiles.UseNew();
45         var exportedNotationFilePath = TemporaryFiles.UseNew();
46         File.WriteAllText(notationFilePath, notation);
47         new LinoImporterCli<TLinkAddress>().Run(notationFilePath, linksStorageFilePath,
    ↪ notationFilePath);
48         new LinoExporterCli<TLinkAddress>().Run(linksStorageFilePath, exportedNotationFilePath,
    ↪ notationFilePath);
49         Assert.Equal(notation, File.ReadAllText(exportedNotationFilePath));
50     }
51 }

```

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

```

1  using Platform.Data.Doublets.Memory;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Doublets.Sequences.Converters;
4  using Platform.Memory;
5  using Xunit;
6  using TLinkAddress = System.UInt64;
7
8  namespace Platform.Data.Doublets.Lino.Tests;
9
10 public class ImporterAndExporterTests
11 {
12     public static ILinks<TLinkAddress> CreateLinks() => CreateLinks(new IO.TemporaryFile());
13
14     public static ILinks<TLinkAddress> CreateLinks(string dbFilename)
15     {
16         var linksConstants = new LinksConstants<TLinkAddress>(true);
17         return new UnitedMemoryLinks<TLinkAddress>(new HeapResizableDirectMemory(),
18             ↪ UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
19             ↪ IndexTreeType.Default);
20     }
21
22     [InlineData("(1: 1 1)")]
23     [InlineData("(1: 1 1)\n(2: 2 2)")]
24     [InlineData("(1: 2 2)")]
25     [InlineData("(1: 2 2)\n(2: 1 1)")]
26     [Theory]
27     public void LinoStorageTest(string notation)
28     {
29         var storage = CreateLinks();
30         var linoStorage = new DefaultLinoStorage<TLinkAddress>(storage);
31         var importer = new LinoImporter<TLinkAddress>(linoStorage);
32         importer.Import(notation);
33         var exporter = new LinoExporter<TLinkAddress>(linoStorage);
34         var exportedLinks = exporter.GetAllLinks();
35         Assert.Equal(notation, exportedLinks);
36     }
37
38     [InlineData("(1: 1 1)")]
39     [InlineData("(1: 1 1)\n(2: 2 2)")]
40     [InlineData("(2: 2 2)")]
41     [InlineData("(1: 2 2)")]
42     [InlineData("(1: 2 2)\n(2: 1 1)")]
43     [InlineData("(1: 2 (3: 3 3))")]
44     [InlineData("(1: 2 (3: 3 3))\n(2: 1 1)")]
45     [InlineData("(son: lovesMama)")]
46     [InlineData("(papa: (lovesMama: loves mama))")]
47     [InlineData("@(papa (lovesMama: loves mama))
48 (son lovesMama)
49 (daughter lovesMama)
50 (all (love mama))")]
51     [Theory]
52     public void LinoDocumentStorageTest(string notation)
53     {
54         var storage = CreateLinks();
55         var linoStorage = new LinoDocumentsStorage<TLinkAddress>(storage, new
56             ↪ BalancedVariantConverter<ulong>(storage));
57         var importer = new LinoImporter<TLinkAddress>(linoStorage);
58         importer.Import(notation);
59         var exporter = new LinoExporter<TLinkAddress>(linoStorage);
60         var exportedLinks = exporter.GetAllLinks();
61         Assert.Equal(notation, exportedLinks);
62     }
63 }

```



## Index

./csharp/Platform.Data.Doublets.Lino.Tests/ImporterAndExporterCliTests.cs, 7  
./csharp/Platform.Data.Doublets.Lino.Tests/ImporterAndExporterTests.cs, 8  
./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, 6  
./csharp/Platform.Data.Doublets.Lino/LinoImporter.cs, 6  
./csharp/Platform.Data.Doublets.Lino/LinoImporterCli.cs, 6