

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, string? documentName)
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(string? documentName)
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, string? documentName);
9     IList<Link> GetLinks(string? documentName);
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, string? documentName)
125     {
126         if (string.IsNullOrEmpty(documentName))
127         {
128             throw new Exception("No document name is passed.");
129         }
130         var sequenceList = new List<TLinkAddress>();
131         for (int i = 0; i < links.Count; i++)
132         {
133             sequenceList.Add(CreateLink(links[i]));
134         }
135         var documentNameSequence = StringToUnicodeSequenceConverter.Convert(documentName);
136         var document = Storage.SearchOrDefault(documentNameSequence, documentNameSequence);
137         if (!_equalityComparer.Equals(default, document))
138         {
139             throw new Exception($"The document with name {documentName} already exists.");
140         }
141         document = Storage.CreateAndUpdate(DocumentMarker, documentNameSequence);
142         Storage.GetOrCreate(document, _listToSequenceConverter.Convert(sequenceList));
143     }
144
145     private TLinkAddress CreateLink(LinoLink link)
146     {
147         if (link.Id != null)
148         {
149             return CreateLinkWithId(link);
150         }
151         var valuesSequence = CreateValuesSequence(link);
152         return Storage.GetOrCreate(LinkWithoutIdMarker, valuesSequence);
153     }
154
155     private TLinkAddress CreateLinkWithId(LinoLink link)

```

```

155 {
156     var currentReference = GetOrCreateReference(link.Id);
157     if (link.Values == null)
158     {
159         return Storage.GetOrCreate(LinkWithIdMarker, currentReference);
160     }
161     var valuesSequence = CreateValuesSequence(link);
162     var idWithValues = Storage.GetOrCreate(currentReference, valuesSequence);
163     return Storage.GetOrCreate(LinkWithIdMarker, idWithValues);
164 }
165
166 private TLinkAddress CreateValuesSequence(LinoLink parent)
167 {
168     var values = new List<TLinkAddress>(parent.Values.Count);
169     for (int i = 0; i < parent.Values.Count; i++)
170     {
171         var currentValue = parent.Values[i];
172         if (currentValue.Values != null)
173         {
174             var valueLink = CreateLink(currentValue);
175             values.Add(valueLink);
176             continue;
177         }
178         var currentValueReference = GetOrCreateReference(currentValue.Id);
179         values.Add(currentValueReference);
180     }
181     return _listToSequenceConverter.Convert(values);
182 }
183
184 private TLinkAddress GetDocument(string documentName)
185 {
186     var documentNameSequence = StringToUnicodeSequenceConverter.Convert(documentName);
187     var document = Storage.SearchOrDefault(DocumentMarker, documentNameSequence);
188     if (_equalityComparer.Equals(default, document))
189     {
190         throw new Exception($"No document in the storage with name {documentName}.");
191     }
192     return document;
193 }
194
195 private TLinkAddress GetDocumentSequence(string documentName)
196 {
197     var document = GetDocument(documentName);
198     return GetDocumentSequence(document);
199 }
200
201 private TLinkAddress GetDocumentSequence(TLinkAddress document)
202 {
203     var any = Storage.Constants.Any;
204     TLinkAddress documentLinksSequence = default;
205     Storage.Each(new Link<TLinkAddress>(any, document, any), link =>
206     {
207         documentLinksSequence = Storage.GetTarget(link);
208         return Storage.Constants.Break;
209     });
210     if (_equalityComparer.Equals(default, documentLinksSequence))
211     {
212         throw new Exception("No links associated with the passed document in the storage.");
213     }
214     return documentLinksSequence;
215 }
216
217 public IList<LinoLink> GetLinks(string? documentName)
218 {
219     if (string.IsNullOrEmpty(documentName))
220     {
221         throw new Exception("No document name is passed.");
222     }
223     var resultLinks = new List<LinoLink>();
224     bool IsElement(TLinkAddress linkIndex)
225     {
226         var source = Storage.GetSource(linkIndex);
227         return _equalityComparer.Equals(LinkWithoutIdMarker, source) |
228             ↳ _equalityComparer.Equals(LinkWithIdMarker, source) ||
229             ↳ Storage.IsPartialPoint(linkIndex);
230     }
231     var document = GetDocument(documentName);

```

```

230     var rightSequenceWalker = new RightSequenceWalker<TLinkAddress>(Storage, new
        ↳ DefaultStack<TLinkAddress>(), IsElement);
231     var documentLinksSequence = GetDocumentSequence(document);
232     var documentLinks = rightSequenceWalker.Walk(documentLinksSequence);
233     foreach (var documentLink in documentLinks)
234     {
235         resultLinks.Add(GetLink(documentLink));
236     }
237     return resultLinks;
238 }
239
240
241 private bool IsLinkOrReferenceOrPartialPoint(TLinkAddress linkIndex)
242 {
243     var link = Storage.GetLink(linkIndex);
244     return IsLink(link) || IsReference(link) || Storage.IsPartialPoint(linkIndex);
245 }
246
247 private LinoLink GetLinkWithId(IList<TLinkAddress> linkWithId)
248 {
249     string id;
250     var values = new List<LinoLink>();
251     var linkStruct = new Link<TLinkAddress>(linkWithId);
252     if (IsReference(linkStruct.Target))
253     {
254         id = ReadReference(linkStruct.Target);
255         return new LinoLink(id);
256     }
257     var rightSequenceWalker = new RightSequenceWalker<TLinkAddress>(Storage, new
        ↳ DefaultStack<TLinkAddress>(), IsLinkOrReferenceOrPartialPoint);
258     using var enumerator = rightSequenceWalker.Walk(linkStruct.Target).GetEnumerator();
259     enumerator.MoveNext();
260     id = ReadReference(enumerator.Current);
261     while (enumerator.MoveNext())
262     {
263         var currentValueStruct = new Link<TLinkAddress>(Storage.GetLink(enumerator.Current));
264         if (IsLink(currentValueStruct))
265         {
266             var value = GetLink(currentValueStruct);
267             values.Add(value);
268         }
269         else if (IsReference(currentValueStruct))
270         {
271             var reference = ReadReference(currentValueStruct);
272             var currentLinoLink = new LinoLink(reference);
273             values.Add(currentLinoLink);
274         }
275     }
276     return new LinoLink(id, values);
277 }
278
279 private LinoLink GetLinkWithoutId(IList<TLinkAddress> linkWithoutId)
280 {
281     var values = new List<LinoLink>();
282     var linkStruct = new Link<TLinkAddress>(linkWithoutId);
283     var rightSequenceWalker = new RightSequenceWalker<TLinkAddress>(Storage, new
        ↳ DefaultStack<TLinkAddress>(), IsLinkOrReferenceOrPartialPoint);
284     foreach (var currentValue in rightSequenceWalker.Walk(linkStruct.Target))
285     {
286         var currentValueStruct = new Link<TLinkAddress>(Storage.GetLink(currentValue));
287         if (IsLink(currentValue))
288         {
289             var value = GetLink(currentValueStruct);
290             values.Add(value);
291         }
292         else if (IsReference(currentValueStruct))
293         {
294             var currentLinoLink = new LinoLink(ReadReference(currentValueStruct));
295             values.Add(currentLinoLink);
296         }
297     }
298     return new LinoLink(null, values);
299 }
300
301 private LinoLink GetLink(TLinkAddress link) => GetLink(Storage.GetLink(link));
302
303 private LinoLink GetLink(IList<TLinkAddress> link)
304 {

```

```

305     var linkStruct = new Link<TLinkAddress>(link);
306     if (IsLinkWithId(linkStruct))
307     {
308         return GetLinkWithId(linkStruct);
309     }
310     if (IsLinkWithoutId(linkStruct))
311     {
312         return GetLinkWithoutId(linkStruct);
313     }
314     throw new Exception("The passed argument is not a link");
315 }
316 }

```

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(string? documentName)
18     {
19         var allLinks = _linoDocumentsStorage.GetLinks(documentName);
20         return allLinks.Format();
21     }
22 }

```

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(documentName);
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, string? documentName)
16    {
17        var linoLinks = _parser.Parse(content);
18        _linoDocumentsStorage.CreateLinks(linoLinks, documentName);
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);
17         var storageFilePath = ConsoleHelpers.GetOrReadArgument(argumentsIndex++, "A path to a
18         ↪ links storage", 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 importer = new LinoImporter<TLinkAddress>(linoStorage);
35         var notation = File.ReadAllText(notationFilePath);
36         importer.Import(notation, documentName);
37     }
38 }
```

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 System;
6 using Xunit;
7 using TLinkAddress = System.UInt64;
8
9 namespace Platform.Data.Doublets.Lino.Tests;
10
11 public class ImporterAndExporterTests
12 {
13     public static ILinks<TLinkAddress> CreateLinks() => CreateLinks(new IO.TemporaryFile());
14
15     public static ILinks<TLinkAddress> CreateLinks(string dbFilename)
16     {
17         var linksConstants = new LinksConstants<TLinkAddress>(true);
18         return new UnitedMemoryLinks<TLinkAddress>(new HeapResizableDirectMemory(),
        ↳ UnitedMemoryLinks<TLinkAddress>.DefaultLinksSizeStep, linksConstants,
        ↳ IndexTreeType.Default);
19     }
20
21     [InlineData("(1: 1 1)")]
22     [InlineData("(1: 1 1)\n(2: 2 2)")]
23     [InlineData("(1: 2 2)")]
24     [InlineData("(1: 2 2)\n(2: 1 1)")]
25     [Theory]
26     public void LinoStorageTest(string notation)
27     {
28         var storage = CreateLinks();
29         var linoStorage = new DefaultLinoStorage<TLinkAddress>(storage);
30         var importer = new LinoImporter<TLinkAddress>(linoStorage);
31         var documentName = Random.RandomHelpers.Default.Next(Int32.MaxValue).ToString();
32         importer.Import(notation, documentName);
33         var exporter = new LinoExporter<TLinkAddress>(linoStorage);
34         var exportedLinks = exporter.GetAllLinks(documentName);
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
        ↳ BalancedVariantConverter<ulong>(storage));
56     var importer = new LinoImporter<TLinkAddress>(linoStorage);
57     var documentName = Random.RandomHelpers.Default.Next(Int32.MaxValue).ToString();
58     importer.Import(notation, documentName);
59     var exporter = new LinoExporter<TLinkAddress>(linoStorage);
60     var exportedLinks = exporter.GetAllLinks(documentName);
61     Assert.Equal(notation, exportedLinks);
62 }
63
64 }

```

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, 6
./csharp/Platform.Data.Doublets.Lino/LinoExporterCli.cs, 6
./csharp/Platform.Data.Doublets.Lino/LinoImporter.cs, 6
./csharp/Platform.Data.Doublets.Lino/LinoImporterCli.cs, 7