

# LinksPlatform's Platform.Data.Doublets.Xml Class Library

## 1.1 ./csharp/Platform.Data.Doublets.Xml/DefaultXmlStorage.cs

```
1 using System.Collections.Generic;
2 using Platform.Numbers;
3 using Platform.Data.Numbers.Raw;
4 using Platform.Data.Doublets;
5 using Platform.Data.Doublets.Sequences.Converters;
6 using Platform.Data.Doublets.Sequences.Frequencies.Cache;
7 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
8 using Platform.Data.Doublets.Sequences.Indexes;
9 using Platform.Data.Doublets.Unicode;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.Xml
14 {
15     /// <summary>
16     /// <para>
17     /// Represents the default xml storage.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     /// <seealso cref="IXmlStorage{TLink}" />
22     public class DefaultXmlStorage<TLink> : IXmlStorage<TLink>
23     {
24         private static readonly TLink _zero = default;
25         private static readonly TLink _one = Arithmetic.Increment(_zero);
26         private readonly StringToUnicodeSequenceConverter<TLink>
27             ↪ _stringToUnicodeSequenceConverter;
28         private readonly ILinks<TLink> _links;
29         private TLink _unicodeSymbolMarker;
30         private TLink _unicodeSequenceMarker;
31         private TLink _elementMarker;
32         private TLink _textElementMarker;
33         private TLink _documentMarker;
34
35         private class Unindex : ISequenceIndex<TLink>
36         {
37             /// <summary>
38             /// <para>
39             /// Determines whether this instance add.
40             /// </para>
41             /// <para></para>
42             /// </summary>
43             /// <param name="sequence">
44             /// <para>The sequence.</para>
45             /// <para></para>
46             /// </param>
47             /// <returns>
48             /// <para>The bool</para>
49             /// <para></para>
50             /// </returns>
51             public bool Add(IList<TLink> sequence) => true;
52
53             /// <summary>
54             /// <para>
55             /// Determines whether this instance might contain.
56             /// </para>
57             /// <para></para>
58             /// </summary>
59             /// <param name="sequence">
60             /// <para>The sequence.</para>
61             /// <para></para>
62             /// </param>
63             /// <returns>
64             /// <para>The bool</para>
65             /// <para></para>
66             /// </returns>
67             public bool MightContain(IList<TLink> sequence) => true;
68         }
69
70         /// <summary>
71         /// <para>
72         /// Initializes a new <see cref="DefaultXmlStorage" /> instance.
73         /// </para>
74         /// <para></para>
75         /// </summary>
76         /// <param name="links">
77         /// <para>A links.</para>
```

```

77     /// <para></para>
78     /// </param>
79     /// <param name="indexSequenceBeforeCreation">
80     /// <para>A index sequence before creation.</para>
81     /// <para></para>
82     /// </param>
83     /// <param name="frequenciesCache">
84     /// <para>A frequencies cache.</para>
85     /// <para></para>
86     /// </param>
87     public DefaultXmlStorage(ILinks<TLink> links, bool indexSequenceBeforeCreation,
88     ↪ LinkFrequenciesCache<TLink> frequenciesCache)
89     {
90         var linkToItsFrequencyNumberConverter = new
91         ↪ FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<TLink>(frequenciesCache);
92         var sequenceToItsLocalElementLevelsConverter = new
93         ↪ SequenceToItsLocalElementLevelsConverter<TLink>(links,
94         ↪ linkToItsFrequencyNumberConverter);
95         var optimalVariantConverter = new OptimalVariantConverter<TLink>(links,
96         ↪ sequenceToItsLocalElementLevelsConverter);
97         InitConstants(links);
98         var charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<TLink>(links,
99         ↪ new AddressToRawNumberConverter<TLink>(), _unicodeSymbolMarker);
100         var index = indexSequenceBeforeCreation ? new
101         ↪ CachedFrequencyIncrementingSequenceIndex<TLink>(frequenciesCache) :
102         ↪ (ISequenceIndex<TLink>)new Unindex();
103         _stringToUnicodeSequenceConverter = new
104         ↪ StringToUnicodeSequenceConverter<TLink>(links, charToUnicodeSymbolConverter,
105         ↪ index, optimalVariantConverter, _unicodeSequenceMarker);
106         _links = links;
107     }
108
109     public DefaultXmlStorage(ILinks<TLink> links, bool indexSequenceBeforeCreation = false) :
110     this(links, indexSequenceBeforeCreation,
111         new LinkFrequenciesCache<TLink>(links,
112         new TotalSequenceSymbolFrequencyCounter<TLink>(links))) { }
113
114     private void InitConstants(ILinks<TLink> links)
115     {
116         var markerIndex = _one;
117         var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
118         _unicodeSymbolMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
119         ↪ markerIndex));
120         _unicodeSequenceMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
121         ↪ markerIndex));
122         _elementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
123         ↪ markerIndex));
124         _textElementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
125         ↪ markerIndex));
126         _documentMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
127         ↪ markerIndex));
128     }
129
130     /// <summary>
131     /// <para>
132     /// Creates the document using the specified name.
133     /// </para>
134     /// <para></para>
135     /// </summary>
136     /// <param name="name">
137     /// <para>The name.</para>
138     /// <para></para>
139     /// </param>
140     /// <returns>
141     /// <para>The link</para>
142     /// <para></para>
143     /// </returns>
144     public TLink CreateDocument(string name) => Create(_documentMarker, name);
145
146     /// <summary>
147     /// <para>
148     /// Creates the element using the specified name.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="name">
153     /// <para>The name.</para>
154     /// <para></para>

```

```

139     /// <para></para>
140     /// </param>
141     /// <returns>
142     /// <para>The link</para>
143     /// <para></para>
144     /// </returns>
145     public TLink CreateElement(string name) => Create(_elementMarker, name);
146
147     /// <summary>
148     /// <para>
149     /// Creates the text element using the specified content.
150     /// </para>
151     /// <para></para>
152     /// </summary>
153     /// <param name="content">
154     /// <para>The content.</para>
155     /// <para></para>
156     /// </param>
157     /// <returns>
158     /// <para>The link</para>
159     /// <para></para>
160     /// </returns>
161     public TLink CreateTextElement(string content) => Create(_textElementMarker, content);
162
163     private TLink Create(TLink marker, string content) => _links.GetOrCreate(marker,
164         ↪ _stringToUnicodeSequenceConverter.Convert(content));
165
166     /// <summary>
167     /// <para>
168     /// Attaches the element to parent using the specified element to attach.
169     /// </para>
170     /// <para></para>
171     /// </summary>
172     /// <param name="elementToAttach">
173     /// <para>The element to attach.</para>
174     /// <para></para>
175     /// </param>
176     /// <param name="parent">
177     /// <para>The parent.</para>
178     /// <para></para>
179     /// </param>
180     public void AttachElementToParent(TLink elementToAttach, TLink parent) =>
181         ↪ _links.GetOrCreate(parent, elementToAttach);
182
183     /// <summary>
184     /// <para>
185     /// Gets the document using the specified name.
186     /// </para>
187     /// <para></para>
188     /// </summary>
189     /// <param name="name">
190     /// <para>The name.</para>
191     /// <para></para>
192     /// </param>
193     /// <returns>
194     /// <para>The link</para>
195     /// <para></para>
196     /// </returns>
197     public TLink GetDocument(string name) => Get(_documentMarker, name);
198
199     /// <summary>
200     /// <para>
201     /// Gets the text element using the specified content.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="content">
206     /// <para>The content.</para>
207     /// <para></para>
208     /// </param>
209     /// <returns>
210     /// <para>The link</para>
211     /// <para></para>
212     /// </returns>
213     public TLink GetTextElement(string content) => Get(_textElementMarker, content);
214     /// <summary>
215     /// <para>
216     /// Gets the element using the specified name.

```

```

215     /// </para>
216     /// <para></para>
217     /// </summary>
218     /// <param name="name">
219     /// <para>The name.</para>
220     /// <para></para>
221     /// </param>
222     /// <returns>
223     /// <para>The link</para>
224     /// <para></para>
225     /// </returns>
226     public TLink GetElement(string name) => Get(_elementMarker, name);
227
228     private TLink Get(TLink marker, string content) => _links.SearchOrDefault(marker,
229         ↪ _stringToUnicodeSequenceConverter.Convert(content));
230
231     /// <summary>
232     /// <para>
233     /// Gets the children using the specified parent.
234     /// </para>
235     /// <para></para>
236     /// </summary>
237     /// <param name="parent">
238     /// <para>The parent.</para>
239     /// <para></para>
240     /// </param>
241     /// <returns>
242     /// <para>The childrens.</para>
243     /// <para></para>
244     /// </returns>
245     public IList<TLink> GetChildren(TLink parent) {
246         List<TLink> childrens = new List<TLink>();
247         _links.Each((link) => {
248             childrens.Add(_links.GetTarget(link));
249             return this._links.Constants.Continue;
250         }, new Link<TLink>(_links.Constants.Any, parent, _links.Constants.Any));
251         return childrens;
252     }
253 }

```

## 1.2 ./csharp/Platform.Data.Doublets.Xml/ ICommandLineInterface.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.Data.Doublets.Xml
4  {
5      /// <summary>
6      /// <para>
7      /// Defines the command line interface.
8      /// </para>
9      /// <para></para>
10     /// </summary>
11     public interface ICommandLineInterface
12     {
13         /// <summary>
14         /// <para>
15         /// Runs the args.
16         /// </para>
17         /// <para></para>
18         /// </summary>
19         /// <param name="args">
20         /// <para>The args.</para>
21         /// <para></para>
22         /// </param>
23         void Run(params string[] args);
24     }
25 }

```

## 1.3 ./csharp/Platform.Data.Doublets.Xml/IXmlStorage.cs

```

1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  using System.Collections.Generic;
4
5  namespace Platform.Data.Doublets.Xml
6  {
7      /// <summary>
8      /// <para>
9      /// Defines the xml storage.

```

```

10  /// </para>
11  /// <para></para>
12  /// </summary>
13  public interface IXmlStorage<TLink>
14  {
15      /// <summary>
16      /// <para>
17      /// Creates the document using the specified name.
18      /// </para>
19      /// <para></para>
20      /// </summary>
21      /// <param name="name">
22      /// <para>The name.</para>
23      /// <para></para>
24      /// </param>
25      /// <returns>
26      /// <para>The link</para>
27      /// <para></para>
28      /// </returns>
29      TLink CreateDocument(string name);
30      /// <summary>
31      /// <para>
32      /// Creates the element using the specified name.
33      /// </para>
34      /// <para></para>
35      /// </summary>
36      /// <param name="name">
37      /// <para>The name.</para>
38      /// <para></para>
39      /// </param>
40      /// <returns>
41      /// <para>The link</para>
42      /// <para></para>
43      /// </returns>
44      TLink CreateElement(string name);
45      /// <summary>
46      /// <para>
47      /// Creates the text element using the specified content.
48      /// </para>
49      /// <para></para>
50      /// </summary>
51      /// <param name="content">
52      /// <para>The content.</para>
53      /// <para></para>
54      /// </param>
55      /// <returns>
56      /// <para>The link</para>
57      /// <para></para>
58      /// </returns>
59      TLink CreateTextElement(string content);
60      /// <summary>
61      /// <para>
62      /// Gets the document using the specified name.
63      /// </para>
64      /// <para></para>
65      /// </summary>
66      /// <param name="name">
67      /// <para>The name.</para>
68      /// <para></para>
69      /// </param>
70      /// <returns>
71      /// <para>The link</para>
72      /// <para></para>
73      /// </returns>
74      TLink GetDocument(string name);
75      /// <summary>
76      /// <para>
77      /// Gets the element using the specified name.
78      /// </para>
79      /// <para></para>
80      /// </summary>
81      /// <param name="name">
82      /// <para>The name.</para>
83      /// <para></para>
84      /// </param>
85      /// <returns>
86      /// <para>The link</para>
87      /// <para></para>

```

```

88     /// </returns>
89     TLink GetElement(string name);
90     /// <summary>
91     /// <para>
92     /// Gets the text element using the specified content.
93     /// </para>
94     /// <para></para>
95     /// </summary>
96     /// <param name="content">
97     /// <para>The content.</para>
98     /// <para></para>
99     /// </param>
100    /// <returns>
101    /// <para>The link</para>
102    /// <para></para>
103    /// </returns>
104    TLink GetTextElement(string content);
105    /// <summary>
106    /// <para>
107    /// Gets the children using the specified parent.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="parent">
112    /// <para>The parent.</para>
113    /// <para></para>
114    /// </param>
115    /// <returns>
116    /// <para>A list of i list t link</para>
117    /// <para></para>
118    /// </returns>
119    IList<TLink> GetChildren(TLink parent);
120    /// <summary>
121    /// <para>
122    /// Attaches the element to parent using the specified element to attach.
123    /// </para>
124    /// <para></para>
125    /// </summary>
126    /// <param name="elementToAttach">
127    /// <para>The element to attach.</para>
128    /// <para></para>
129    /// </param>
130    /// <param name="parent">
131    /// <para>The parent.</para>
132    /// <para></para>
133    /// </param>
134    void AttachElementToParent(TLink elementToAttach, TLink parent);
135 }
136 }

```

#### 1.4 ./csharp/Platform.Data.Doublets.Xml/XmlElementContext.cs

```

1  using System.Collections.Generic;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.Data.Doublets.Xml
6  {
7      /// <summary>
8      /// <para>
9      /// Represents the xml element context.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     internal class XmlElementContext
14     {
15         /// <summary>
16         /// <para>
17         /// The children names counts.
18         /// </para>
19         /// <para></para>
20         /// </summary>
21         public readonly Dictionary<string, int> ChildrenNamesCounts;
22         /// <summary>
23         /// <para>
24         /// The total children.
25         /// </para>
26         /// <para></para>
27         /// </summary>

```

```

28     public int TotalChildren;
29
30     /// <summary>
31     /// <para>
32     /// Initializes a new <see cref="XmlElementContext"/> instance.
33     /// </para>
34     /// <para></para>
35     /// </summary>
36     public XmlElementContext() => ChildrenNamesCounts = new Dictionary<string, int>();
37
38     /// <summary>
39     /// <para>
40     /// Increments the child name count using the specified name.
41     /// </para>
42     /// <para></para>
43     /// </summary>
44     /// <param name="name">
45     /// <para>The name.</para>
46     /// <para></para>
47     /// </param>
48     public void IncrementChildNameCount(string name)
49     {
50         if (ChildrenNamesCounts.TryGetValue(name, out int count))
51         {
52             ChildrenNamesCounts[name] = count + 1;
53         }
54         else
55         {
56             ChildrenNamesCounts[name] = 0;
57         }
58         TotalChildren++;
59     }
60 }
61 }

```

## 1.5 ./csharp/Platform.Data.Doublets.Xml/XmlElementCounter.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Threading;
4  using System.Threading.Tasks;
5  using System.Xml;
6  using System.Linq;
7  using Platform.Exceptions;
8  using Platform.IO;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Xml
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the xml element counter.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     public class XmlElementCounter
21     {
22         /// <summary>
23         /// <para>
24         /// Initializes a new <see cref="XmlElementCounter"/> instance.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         public XmlElementCounter() { }
29
30         /// <summary>
31         /// <para>
32         /// Counts the file.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="file">
37         /// <para>The file.</para>
38         /// <para></para>
39         /// </param>
40         /// <param name="elementName">
41         /// <para>The element name.</para>
42         /// <para></para>
43         /// </param>

```

```

44  /// <param name="token">
45  /// <para>The token.</para>
46  /// <para></para>
47  /// </param>
48  public Task Count(string file, string elementName, CancellationToken token)
49  {
50      return Task.Factory.StartNew(() =>
51      {
52          try
53          {
54              var context = new RootElementContext();
55              using (var reader = XmlReader.Create(file))
56              {
57                  Count(reader, elementName, token, context);
58              }
59              Console.WriteLine($"Total elements with specified name:
60                  ↪ {context.TotalElements}, total content length:
61                  ↪ {context.TotalContentsLength}.");
62          }
63          catch (Exception ex)
64          {
65              Console.WriteLine(ex.ToStringWithAllInnerExceptions());
66          }
67      }, token);
68  }
69  /// <summary>
70  /// <para>
71  /// Counts the reader.
72  /// </para>
73  /// <para></para>
74  /// </summary>
75  /// <param name="reader">
76  /// <para>The reader.</para>
77  /// <para></para>
78  /// </param>
79  /// <param name="elementNameToCount">
80  /// <para>The element name to count.</para>
81  /// <para></para>
82  /// </param>
83  /// <param name="token">
84  /// <para>The token.</para>
85  /// <para></para>
86  /// </param>
87  /// <param name="context">
88  /// <para>The context.</para>
89  /// <para></para>
90  /// </param>
91  private void Count(XmlReader reader, string elementNameToCount, CancellationToken token,
92  ↪ XmlElementContext context)
93  {
94      var rootContext = (RootElementContext)context;
95      var parentContexts = new Stack<XmlElementContext>();
96      var elements = new Stack<string>(); // Path
97      // TODO: If path was loaded previously, skip it.
98      while (reader.Read())
99      {
100          if (token.IsCancellationRequested)
101          {
102              return;
103          }
104          switch (reader.NodeType)
105          {
106              case XmlNodeType.Element:
107                  var elementName = reader.Name;
108                  context.IncrementChildNameCount(elementName);
109                  elementName =
110                  ↪ $"{elementName}[{context.ChildrenNamesCounts[elementName]}]";
111                  if (!reader.IsEmptyElement)
112                  {
113                      elements.Push(elementName);
114                      ConsoleHelpers.Debug("{0} starting...", elements.Count <= 20 ?
115                      ↪ ToXPath(elements) : elementName); // XPath
116                      parentContexts.Push(context);
117                      context = new XmlElementContext();
118                  }
119                  else
120                  {
121

```



```

117         ConsoleHelpers.Debug("{0} finished.", elementName);
118     }
119     break;
120
121     case XmlNodeType.EndElement:
122         ConsoleHelpers.Debug("{0} finished.", elements.Count <= 20 ?
            ↳ ToXPath(elements) : elements.Peek()); // XPath
123         var topElement = elements.Pop();
124         // Restoring scope
125         context = parentContexts.Pop();
126         if (topElement.StartsWith(elementNameToCount))
127         {
128             rootContext.TotalElements++;
129             // TODO: Check for 0x00 part/symbol at 198102797 line and 13
130             ↳ position.
131             //if (rootContext.TotalPages > 3490000)
132             //    selfCancel = true;
133             if (context.ChildrenNamesCounts[elementNameToCount] % 10000 == 0)
134             {
135                 Console.WriteLine(topElement);
136             }
137         }
138         break;
139
140     case XmlNodeType.Text:
141         ConsoleHelpers.Debug("Starting text element...");
142         var content = reader.Value;
143         rootContext.TotalContentsLength += (ulong)content.Length;
144         ConsoleHelpers.Debug($"Content length is: {content.Length}");
145         ConsoleHelpers.Debug("Text element finished.");
146         break;
147     }
148 }
149
150 /// <summary>
151 /// <para>
152 /// Returns the x path using the specified path.
153 /// </para>
154 /// <para></para>
155 /// </summary>
156 /// <param name="path">
157 /// <para>The path.</para>
158 /// <para></para>
159 /// </param>
160 /// <returns>
161 /// <para>The string</para>
162 /// <para></para>
163 /// </returns>
164 private string ToXPath(Stack<string> path) => string.Join("/", path.Reverse());
165
166 /// <summary>
167 /// <para>
168 /// Represents the root element context.
169 /// </para>
170 /// <para></para>
171 /// </summary>
172 /// <seealso cref="XmlElementContext"/>
173 private class RootElementContext : XmlElementContext
174 {
175     /// <summary>
176     /// <para>
177     /// The total elements.
178     /// </para>
179     /// <para></para>
180     /// </summary>
181     public ulong TotalElements;
182     /// <summary>
183     /// <para>
184     /// The total contents length.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     public ulong TotalContentsLength;
189 }
190 }
191 }

```

## 1.6 ./csharp/Platform.Data.Doublets.Xml/XmlElementCounterCLI.cs

```
1 using System;
2 using System.IO;
3 using Platform.IO;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.Data.Doublets.Xml
8 {
9     /// <summary>
10    /// <para>
11    /// Represents the xml element counter cli.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    /// <seealso cref="ICommandLineInterface"/>
16    public class XmlElementCounterCLI : ICommandLineInterface
17    {
18        /// <summary>
19        /// <para>
20        /// Runs the args.
21        /// </para>
22        /// <para></para>
23        /// </summary>
24        /// <param name="args">
25        /// <para>The args.</para>
26        /// <para></para>
27        /// </param>
28        public void Run(params string[] args)
29        {
30            var file = ConsoleHelpers.GetOrReadArgument(0, "Xml file", args);
31            var elementName = ConsoleHelpers.GetOrReadArgument(1, "Element name to count", args);
32            if (!File.Exists(file))
33            {
34                Console.WriteLine("Entered xml file does not exists.");
35            }
36            else if (string.IsNullOrEmpty(elementName))
37            {
38                Console.WriteLine("Entered element name is empty.");
39            }
40            else
41            {
42                using (var cancellation = new ConsoleCancellation())
43                {
44                    Console.WriteLine("Press CTRL+C to stop.");
45                    new XmlElementCounter().Count(file, elementName, cancellation.Token).Wait();
46                }
47            }
48        }
49    }
50 }
```

## 1.7 ./csharp/Platform.Data.Doublets.Xml/XmlExporter.cs

```
1 using System;
2 using System.Linq;
3 using System.Collections.Generic;
4 using System.Threading;
5 using System.Threading.Tasks;
6 using System.Xml;
7 using Platform.Exceptions;
8
9 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.Data.Doublets.Xml
12 {
13     /// <summary>
14     /// <para>
15     /// Represents the xml exporter.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     class XmlExporter<TLink>
20     {
21         /// <summary>
22         /// <para>
23         /// The storage.
24         /// </para>
25         /// <para></para>
26         /// </summary>
```

```

27     private readonly IXmlStorage<TLink> _storage;
28
29     /// <summary>
30     /// <para>
31     /// Initializes a new <see cref="XmlExporter"/> instance.
32     /// </para>
33     /// <para></para>
34     /// </summary>
35     /// <param name="storage">
36     /// <para>A storage.</para>
37     /// <para></para>
38     /// </param>
39     public XmlExporter(IXmlStorage<TLink> storage) => _storage = storage;
40
41     /// <summary>
42     /// <para>
43     /// Exports the document name.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="documentName">
48     /// <para>The document name.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="fileName">
52     /// <para>The file name.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="token">
56     /// <para>The token.</para>
57     /// <para></para>
58     /// </param>
59     public Task Export(string documentName, string fileName, CancellationToken token)
60     {
61         return Task.Factory.StartNew(() =>
62         {
63             try
64             {
65                 var document = _storage.GetDocument(documentName);
66                 using (var writer = XmlWriter.Create(fileName))
67                 {
68                     Write(writer, token, new ElementContext(document));
69                 }
70             }
71             catch (Exception ex)
72             {
73                 Console.WriteLine(ex.ToStringWithAllInnerExceptions());
74             }
75             }, token);
76     }
77
78     private void Write(XmlWriter writer, CancellationToken token, ElementContext context)
79     {
80         var parentContexts = new Stack<ElementContext>();
81         var elements = new Stack<string>(); // Path
82         // TODO: If path was loaded previously, skip it.
83         foreach (TLink lvl in _storage.GetChildren(parent: context.Parent))
84         {
85             Write(writer: writer, token: token, context: new ElementContext(lvl));
86         }
87     }
88
89     private class ElementContext : XmlElementContext
90     {
91         public readonly TLink Parent;
92         public ElementContext(TLink parent) => Parent = parent;
93     }
94
95 }
96

```

## 1.8 ./csharp/Platform.Data.Doublets.Xml/XmlExporterCLI.cs

```

1  using System;
2  using System.IO;
3  using Platform.IO;
4  using Platform.Data.Doublets.Memory.United.Generic;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7

```

```

8 namespace Platform.Data.Doublets.Xml
9 {
10     public class XmlExporterCLI : ICommandLineInterface
11     {
12         public void Run(params string[] args)
13         {
14             var linksFile = ConsoleHelpers.GetOrReadArgument(0, "Links file", args);
15             var exportFile = ConsoleHelpers.GetOrReadArgument(1, "Xml file", args);
16
17             if (File.Exists(exportFile))
18             {
19                 Console.WriteLine("Entered xml file does already exists.");
20             }
21             else
22             {
23                 using (var cancellation = new ConsoleCancellation())
24                 {
25                     using (var memoryAdapter = new UnitedMemoryLinks<uint>(linksFile))
26                     {
27                         {
28                             Console.WriteLine("Press CTRL+C to stop.");
29                             var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesRe
30                                 ↪ solution();
31                             if (cancellation.NotRequested)
32                             {
33                                 var storage = new DefaultXmlStorage<uint>(links);
34                                 var exporter = new XmlExporter<uint>(storage);
35                                 exporter.Export(linksFile, exportFile,
36                                     ↪ cancellation.Token).Wait();
37                             }
38                         }
39                     }
40                 }
41             }
42         }
43     }
44 }

```

## 1.9 ./csharp/Platform.Data.Doublets.Xml/XMLImporter.cs

```

1 using System;
2 using System.Linq;
3 using System.Collections.Generic;
4 using System.Threading;
5 using System.Threading.Tasks;
6 using System.Xml;
7 using Platform.Exceptions;
8 using Platform.Collections;
9 using Platform.IO;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.Xml {
14     /// <summary>
15     /// <para>
16     /// Represents the xml importer.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     public class XmlImporter<TLink>
21     {
22         /// <summary>
23         /// <para>
24         /// The storage.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         private readonly IXmlStorage<TLink> _storage;
29
30         /// <summary>
31         /// <para>
32         /// Initializes a new <see cref="XmlImporter"/> instance.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         /// <param name="storage">
37         /// <para>A storage.</para>
38         /// <para></para>
39         /// </param>
40         public XmlImporter(IXmlStorage<TLink> storage) => _storage = storage;
41     }
42 }

```

```

41
42     /// <summary>
43     /// <para>
44     /// Imports the file.
45     /// </para>
46     /// <para></para>
47     /// </summary>
48     /// <param name="file">
49     /// <para>The file.</para>
50     /// <para></para>
51     /// </param>
52     /// <param name="token">
53     /// <para>The token.</para>
54     /// <para></para>
55     /// </param>
56 public Task Import(string file, CancellationToken token)
57 {
58     return Task.Factory.StartNew(() =>
59     {
60         try
61         {
62             var document = _storage.CreateDocument(file);
63
64             using (var reader = XmlReader.Create(file))
65             {
66                 Read(reader, token, new ElementContext(document));
67             }
68         }
69         catch (Exception ex)
70         {
71             Console.WriteLine(ex.ToStringWithAllInnerExceptions());
72         }
73     }, token);
74 }
75
76
77     /// <summary>
78     /// <para>
79     /// Reads the reader.
80     /// </para>
81     /// <para></para>
82     /// </summary>
83     /// <param name="reader">
84     /// <para>The reader.</para>
85     /// <para></para>
86     /// </param>
87     /// <param name="token">
88     /// <para>The token.</para>
89     /// <para></para>
90     /// </param>
91     /// <param name="context">
92     /// <para>The context.</para>
93     /// <para></para>
94     /// </param>
95 private void Read(XmlReader reader, CancellationToken token, ElementContext context)
96 {
97     var parentContexts = new Stack<ElementContext>();
98     var elements = new Stack<string>(); // Path
99     // TODO: If path was loaded previously, skip it.
100    while (reader.Read())
101    {
102        if (token.IsCancellationRequested)
103        {
104            return;
105        }
106        switch (reader.NodeType)
107        {
108            case XmlNodeType.Element:
109                var elementName = reader.Name;
110                context.IncrementChildNameCount(elementName);
111                elementName =
112                    ↳ $"{{{elementName}}}[{context.ChildrenNamesCounts[elementName]}]";
113                if (!reader.IsEmptyElement)
114                {
115                    elements.Push(elementName);
116                    ConsoleHelpers.Debug("{0} starting...", elements.Count <= 20 ?
117                        ↳ ToXPath(elements) : elementName); // XPath
118                    var element = _storage.CreateElement(name: elementName);

```

```

117         parentContexts.Push(context);
118         _storage.AttachElementToParent(elementToAttach: element, parent:
            ↪ context.Parent);
119         context = new ElementContext(element);
120     }
121     else
122     {
123         ConsoleHelpers.Debug("{0} finished.", elementName);
124     }
125     break;
126 case XmlNodeType.EndElement:
127     ConsoleHelpers.Debug("{0} finished.", elements.Count <= 20 ?
            ↪ ToXPath(elements) : elements.Peek()); // XPath
128     elements.Pop();
129     // Restoring scope
130     context = parentContexts.Pop();
131     if (elements.Count == 1)
132     {
133         if (context.TotalChildren % 10 == 0)
134             Console.WriteLine(context.TotalChildren);
135     }
136     break;
137 case XmlNodeType.Text:
138     ConsoleHelpers.Debug("Starting text element...");
139     var content = reader.Value;
140     ConsoleHelpers.Debug("Content: {0}-{1}", content.Truncate(50),
            ↪ content.Length >= 50 ? "... " : "");
141     var textElement = _storage.CreateTextElement(content: content);
142     _storage.AttachElementToParent(textElement, context.Parent);
143     ConsoleHelpers.Debug("Text element finished.");
144     break;
145 }
146 }
147 }
148
149 /// <summary>
150 /// <para>
151 /// Returns the x path using the specified path.
152 /// </para>
153 /// <para></para>
154 /// </summary>
155 /// <param name="path">
156 /// <para>The path.</para>
157 /// <para></para>
158 /// </param>
159 /// <returns>
160 /// <para>The string</para>
161 /// <para></para>
162 /// </returns>
163 private string ToXPath(Stack<string> path) => string.Join("/", path.Reverse());
164
165 /// <summary>
166 /// <para>
167 /// Represents the element context.
168 /// </para>
169 /// <para></para>
170 /// </summary>
171 /// <seealso cref="XmlElementContext"/>
172 private class ElementContext : XmlElementContext
173 {
174     /// <summary>
175     /// <para>
176     /// The parent.
177     /// </para>
178     /// <para></para>
179     /// </summary>
180     public readonly TLink Parent;
181
182     /// <summary>
183     /// <para>
184     /// Initializes a new <see cref="ElementContext"/> instance.
185     /// </para>
186     /// <para></para>
187     /// </summary>
188     /// <param name="parent">
189     /// <para>A parent.</para>
190     /// <para></para>
191     /// </param>

```

```

192         public ElementContext(TLink parent)=> Parent = parent;
193     }
194 }
195 }

```

### 1.10 ./csharp/Platform.Data.Doublets.Xml/XmlImporterCLI.cs

```

1  using System;
2  using System.IO;
3  using Platform.IO;
4  using Platform.Data.Doublets.Memory.United.Generic;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Xml
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the xml importer cli.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="ICommandLineInterface"/>
17     public class XmlImporterCLI : ICommandLineInterface
18     {
19         /// <summary>
20         /// <para>
21         /// Runs the args.
22         /// </para>
23         /// <para></para>
24         /// </summary>
25         /// <param name="args">
26         /// <para>The args.</para>
27         /// <para></para>
28         /// </param>
29         public void Run(params string[] args)
30         {
31             var linksFile = ConsoleHelpers.GetOrReadArgument(0, "Links file", args);
32             var file = ConsoleHelpers.GetOrReadArgument(1, "Xml file", args);
33
34             if (!File.Exists(file))
35             {
36                 Console.WriteLine("Entered xml file does not exists.");
37             }
38             else
39             {
40                 //const long gb32 = 34359738368;
41
42                 using (var cancellation = new ConsoleCancellation())
43                 using (var memoryAdapter = new UnitedMemoryLinks<uint>(linksFile))
44                 //using (var memoryAdapter = new UInt64UnitedMemoryLinks(linksFile, gb32))
45                 //using (var links = new UInt64Links(memoryAdapter))
46                 {
47                     Console.WriteLine("Press CTRL+C to stop.");
48                     var links =
49                     ↪ memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
50                     var indexer = new XmlIndexer<uint>(links);
51                     var indexingImporter = new XmlImporter<uint>(indexer);
52                     indexingImporter.Import(file, cancellation.Token).Wait();
53                     if (cancellation.NotRequested)
54                     {
55                         var cache = indexer.Cache;
56                         //var counter = new TotalSequenceSymbolFrequencyCounter<uint>(links);
57                         //var cache = new LinkFrequenciesCache<uint>(links, counter);
58                         Console.WriteLine("Frequencies cache ready.");
59                         var storage = new DefaultXmlStorage<uint>(links, false, cache);
60                         var importer = new XmlImporter<uint>(storage);
61                         importer.Import(file, cancellation.Token).Wait();
62                     }
63                 }
64             }
65         }
66     }

```

### 1.11 ./csharp/Platform.Data.Doublets.Xml/XmlIndexer.cs

```

1  using System.Collections.Generic;
2  using Platform.Numbers;
3  using Platform.Data.Numbers.Raw;
4  using Platform.Data.Doublets;

```

```

5 using Platform.Data.Doublets.Sequences.Frequencies.Cache;
6 using Platform.Data.Doublets.Sequences.Frequencies.Counters;
7 using Platform.Data.Doublets.Sequences.Indexes;
8 using Platform.Data.Doublets.Unicode;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublets.Xml
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the xml indexer.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="IXmlStorage{TLink}" />
21     public class XmlIndexer<TLink> : IXmlStorage<TLink>
22     {
23         /// <summary>
24         /// <para>
25         /// The zero.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         private static readonly TLink _zero = default;
30         /// <summary>
31         /// <para>
32         /// The zero.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         private static readonly TLink _one = Arithmetic.Increment(_zero);
37
38         /// <summary>
39         /// <para>
40         /// The index.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         private readonly CachedFrequencyIncrementingSequenceIndex<TLink> _index;
45         /// <summary>
46         /// <para>
47         /// The char to unicode symbol converter.
48         /// </para>
49         /// <para></para>
50         /// </summary>
51         private readonly CharToUnicodeSymbolConverter<TLink> _charToUnicodeSymbolConverter;
52         /// <summary>
53         /// <para>
54         /// The unicode symbol marker.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         private TLink _unicodeSymbolMarker;
59         /// <summary>
60         /// <para>
61         /// The null constant.
62         /// </para>
63         /// <para></para>
64         /// </summary>
65         private readonly TLink _nullConstant;
66
67         /// <summary>
68         /// <para>
69         /// Gets the cache value.
70         /// </para>
71         /// <para></para>
72         /// </summary>
73         public LinkFrequenciesCache<TLink> Cache { get; }
74
75         /// <summary>
76         /// <para>
77         /// Initializes a new <see cref="XmlIndexer" /> instance.
78         /// </para>
79         /// <para></para>
80         /// </summary>
81         /// <param name="links">
82         /// <para>A links.</para>
83         /// <para></para>

```



```

84  /// </param>
85  public XmlIndexer(ILinks<TLink> links)
86  {
87      _nullConstant = links.Constants.Null;
88      var totalSequenceSymbolFrequencyCounter = new
89          ↪ TotalSequenceSymbolFrequencyCounter<TLink>(links);
90      Cache = new LinkFrequenciesCache<TLink>(links, totalSequenceSymbolFrequencyCounter);
91      _index = new CachedFrequencyIncrementingSequenceIndex<TLink>(Cache);
92      var addressToRawNumberConverter = new AddressToRawNumberConverter<TLink>();
93      InitConstants(links);
94      _charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<TLink>(links,
95          ↪ addressToRawNumberConverter, _unicodeSymbolMarker);
96  }
97
98  /// <summary>
99  /// <para>
100  ///     Inits the constants using the specified links.
101  /// </para>
102  /// </summary>
103  /// <param name="links">
104  ///     <para>The links.</para>
105  /// </param>
106  private void InitConstants(ILinks<TLink> links)
107  {
108      var markerIndex = _one;
109      var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
110      _unicodeSymbolMarker = links.GetOrCreate(meaningRoot,
111          ↪ Arithmetic.Increment(markerIndex));
112      _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
113      _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
114      _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
115      _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
116  }
117
118  /// <summary>
119  /// <para>
120  ///     Attaches the element to parent using the specified element to attach.
121  /// </para>
122  /// </summary>
123  /// <param name="elementToAttach">
124  ///     <para>The element to attach.</para>
125  /// </param>
126  /// <param name="parent">
127  ///     <para>The parent.</para>
128  /// </param>
129  public void AttachElementToParent(TLink elementToAttach, TLink parent)
130  {
131  }
132
133  /// <summary>
134  /// <para>
135  ///     Returns the elements using the specified string.
136  /// </para>
137  /// </summary>
138  /// <param name="@string">
139  ///     <para>The string.</para>
140  /// </param>
141  /// <returns>
142  ///     <para>The elements.</para>
143  /// </returns>
144  public IList<TLink> ToElements(string @string)
145  {
146      var elements = new TLink[@string.Length];
147      for (int i = 0; i < @string.Length; i++)
148      {
149          elements[i] = _charToUnicodeSymbolConverter.Convert(@string[i]);
150      }
151      return elements;
152  }
153
154  }
155
156  }
157
158  }

```

```

159     /// <summary>
160     /// <para>
161     /// Creates the document using the specified name.
162     /// </para>
163     /// <para></para>
164     /// </summary>
165     /// <param name="name">
166     /// <para>The name.</para>
167     /// <para></para>
168     /// </param>
169     /// <returns>
170     /// <para>The null constant.</para>
171     /// <para></para>
172     /// </returns>
173     public TLink CreateDocument(string name)
174     {
175         _index.Add(ToElements(name));
176         return _nullConstant;
177     }
178
179     /// <summary>
180     /// <para>
181     /// Creates the element using the specified name.
182     /// </para>
183     /// <para></para>
184     /// </summary>
185     /// <param name="name">
186     /// <para>The name.</para>
187     /// <para></para>
188     /// </param>
189     /// <returns>
190     /// <para>The null constant.</para>
191     /// <para></para>
192     /// </returns>
193     public TLink CreateElement(string name)
194     {
195         _index.Add(ToElements(name));
196         return _nullConstant;
197     }
198
199     /// <summary>
200     /// <para>
201     /// Creates the text element using the specified content.
202     /// </para>
203     /// <para></para>
204     /// </summary>
205     /// <param name="content">
206     /// <para>The content.</para>
207     /// <para></para>
208     /// </param>
209     /// <returns>
210     /// <para>The null constant.</para>
211     /// <para></para>
212     /// </returns>
213     public TLink CreateTextElement(string content)
214     {
215         _index.Add(ToElements(content));
216         return _nullConstant;
217     }
218
219     /// <summary>
220     /// <para>
221     /// Gets the document using the specified name.
222     /// </para>
223     /// <para></para>
224     /// </summary>
225     /// <param name="name">
226     /// <para>The name.</para>
227     /// <para></para>
228     /// </param>
229     /// <exception cref="System.NotImplementedException">
230     /// <para></para>
231     /// <para></para>
232     /// </exception>
233     /// <returns>
234     /// <para>The link</para>
235     /// <para></para>
236     /// </returns>

```

```

237 public TLink GetDocument(string name)
238 {
239     throw new System.NotImplementedException();
240 }
241
242 /// <summary>
243 /// <para>
244 /// Gets the element using the specified name.
245 /// </para>
246 /// <para></para>
247 /// </summary>
248 /// <param name="name">
249 /// <para>The name.</para>
250 /// <para></para>
251 /// </param>
252 /// <exception cref="System.NotImplementedException">
253 /// <para></para>
254 /// <para></para>
255 /// </exception>
256 /// <returns>
257 /// <para>The link</para>
258 /// <para></para>
259 /// </returns>
260 public TLink GetElement(string name)
261 {
262     throw new System.NotImplementedException();
263 }
264
265 /// <summary>
266 /// <para>
267 /// Gets the text element using the specified content.
268 /// </para>
269 /// <para></para>
270 /// </summary>
271 /// <param name="content">
272 /// <para>The content.</para>
273 /// <para></para>
274 /// </param>
275 /// <exception cref="System.NotImplementedException">
276 /// <para></para>
277 /// <para></para>
278 /// </exception>
279 /// <returns>
280 /// <para>The link</para>
281 /// <para></para>
282 /// </returns>
283 public TLink GetTextElement(string content)
284 {
285     throw new System.NotImplementedException();
286 }
287
288 /// <summary>
289 /// <para>
290 /// Gets the children using the specified parent.
291 /// </para>
292 /// <para></para>
293 /// </summary>
294 /// <param name="parent">
295 /// <para>The parent.</para>
296 /// <para></para>
297 /// </param>
298 /// <exception cref="System.NotImplementedException">
299 /// <para></para>
300 /// <para></para>
301 /// </exception>
302 /// <returns>
303 /// <para>A list of i list t link</para>
304 /// <para></para>
305 /// </returns>
306 public IList<IList<TLink>> GetChildren(TLink parent)
307 {
308     throw new System.NotImplementedException();
309 }
310 }
311 }

```

## Index

./csharp/Platform.Data.Doublets.Xml/DefaultXmlStorage.cs, 1  
./csharp/Platform.Data.Doublets.Xml/ ICommandLineInterface.cs, 4  
./csharp/Platform.Data.Doublets.Xml/IXmlStorage.cs, 4  
./csharp/Platform.Data.Doublets.Xml/XmlElementContext.cs, 6  
./csharp/Platform.Data.Doublets.Xml/XmlElementCounter.cs, 7  
./csharp/Platform.Data.Doublets.Xml/XmlElementCounterCLI.cs, 9  
./csharp/Platform.Data.Doublets.Xml/XmlExporter.cs, 10  
./csharp/Platform.Data.Doublets.Xml/XmlExporterCLI.cs, 11  
./csharp/Platform.Data.Doublets.Xml/XmlImporter.cs, 12  
./csharp/Platform.Data.Doublets.Xml/XmlImporterCLI.cs, 15  
./csharp/Platform.Data.Doublets.Xml/XmlIndexer.cs, 15