

LinksPlatform's Platform.Data.Doublents.Xml Class Library

1.1 ./csharp/Platform.Data.Doublents.Xml/DefaultXmlStorage.cs

```
1 using System.Collections.Generic;
2 using Platform.Numbers;
3 using Platform.Data.Numbers.Raw;
4 using Platform.Data.Doublents;
5 using Platform.Data.Doublents.Sequences.Converters;
6 using Platform.Data.Doublents.Sequences.Frequencies.Cache;
7 using Platform.Data.Doublents.Sequences.Indexes;
8 using Platform.Data.Doublents.Unicode;
9
10 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
11
12 namespace Platform.Data.Doublents.Xml
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the default xml storage.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     /// <seealso cref="IXmlStorage{TLink}" />
21     public class DefaultXmlStorage<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 string to unicode sequence converter.
41         /// </para>
42         /// <para></para>
43         /// </summary>
44         private readonly StringToUnicodeSequenceConverter<TLink>
45         ↪ _stringToUnicodeSequenceConverter;
46         /// <summary>
47         /// <para>
48         /// The links.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         private readonly ILinks<TLink> _links;
53         /// <summary>
54         /// <para>
55         /// The unicode symbol marker.
56         /// </para>
57         /// <para></para>
58         /// </summary>
59         private TLink _unicodeSymbolMarker;
60         /// <summary>
61         /// <para>
62         /// The unicode sequence marker.
63         /// </para>
64         /// <para></para>
65         /// </summary>
66         private TLink _unicodeSequenceMarker;
67         /// <summary>
68         /// <para>
69         /// The element marker.
70         /// </para>
71         /// <para></para>
72         /// </summary>
73         private TLink _elementMarker;
74         /// <summary>
75         /// <para>
76         /// The text element marker.
77         /// </para>
```

```

77     /// <para></para>
78     /// </summary>
79     private TLink _textElementMarker;
80     /// <summary>
81     /// <para>
82     /// The document marker.
83     /// </para>
84     /// <para></para>
85     /// </summary>
86     private TLink _documentMarker;
87
88     /// <summary>
89     /// <para>
90     /// Represents the unindex.
91     /// </para>
92     /// <para></para>
93     /// </summary>
94     /// <seealso cref="ISequenceIndex{TLink}"/>
95     private class Unindex : ISequenceIndex<TLink>
96     {
97         /// <summary>
98         /// <para>
99         /// Determines whether this instance add.
100        /// </para>
101        /// <para></para>
102        /// </summary>
103        /// <param name="sequence">
104        /// <para>The sequence.</para>
105        /// <para></para>
106        /// </param>
107        /// <returns>
108        /// <para>The bool</para>
109        /// <para></para>
110        /// </returns>
111        public bool Add(IList<TLink> sequence) => true;
112        /// <summary>
113        /// <para>
114        /// Determines whether this instance might contain.
115        /// </para>
116        /// <para></para>
117        /// </summary>
118        /// <param name="sequence">
119        /// <para>The sequence.</para>
120        /// <para></para>
121        /// </param>
122        /// <returns>
123        /// <para>The bool</para>
124        /// <para></para>
125        /// </returns>
126        public bool MightContain(IList<TLink> sequence) => true;
127    }
128
129    /// <summary>
130    /// <para>
131    /// Initializes a new <see cref="DefaultXmlStorage"/> instance.
132    /// </para>
133    /// <para></para>
134    /// </summary>
135    /// <param name="links">
136    /// <para>A links.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="indexSequenceBeforeCreation">
140    /// <para>A index sequence before creation.</para>
141    /// <para></para>
142    /// </param>
143    /// <param name="frequenciesCache">
144    /// <para>A frequencies cache.</para>
145    /// <para></para>
146    /// </param>
147    public DefaultXmlStorage(ILinks<TLink> links, bool indexSequenceBeforeCreation,
148        ↪ LinkFrequenciesCache<TLink> frequenciesCache)
149    {
150        var linkToItsFrequencyNumberConverter = new
151            ↪ FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<TLink>(frequenciesCache);
152        var sequenceToItsLocalElementLevelsConverter = new
153            ↪ SequenceToItsLocalElementLevelsConverter<TLink>(links,
154            ↪ linkToItsFrequencyNumberConverter);

```

```

151     var optimalVariantConverter = new OptimalVariantConverter<TLink>(links,
152         ↳ sequenceToItsLocalElementLevelsConverter);
153     InitConstants(links);
154     var charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<TLink>(links,
155         ↳ new AddressToRawNumberConverter<TLink>(), _unicodeSymbolMarker);
156     var index = indexSequenceBeforeCreation ? new
157         ↳ CachedFrequencyIncrementingSequenceIndex<TLink>(frequenciesCache) :
158         ↳ (ISequenceIndex<TLink>)new Unindex();
159     _stringToUnicodeSequenceConverter = new
160         ↳ StringToUnicodeSequenceConverter<TLink>(links, charToUnicodeSymbolConverter,
161         ↳ index, optimalVariantConverter, _unicodeSequenceMarker);
162     _links = links;
163 }
164
165 /// <summary>
166 /// <para>
167 ///     Inits the constants using the specified links.
168 /// </para>
169 /// </summary>
170 /// <param name="links">
171 /// <para>The links.</para>
172 /// </param>
173 private void InitConstants(ILinks<TLink> links)
174 {
175     var markerIndex = _one;
176     var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
177     _unicodeSymbolMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
178         ↳ markerIndex));
179     _unicodeSequenceMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
180         ↳ markerIndex));
181     _elementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
182         ↳ markerIndex));
183     _textElementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
184         ↳ markerIndex));
185     _documentMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
186         ↳ markerIndex));
187 }
188
189 /// <summary>
190 /// <para>
191 ///     Creates the document using the specified name.
192 /// </para>
193 /// </summary>
194 /// <param name="name">
195 /// <para>The name.</para>
196 /// </param>
197 /// <returns>
198 /// <para>The link</para>
199 /// </returns>
200 public TLink CreateDocument(string name) => Create(_documentMarker, name);
201
202 /// <summary>
203 /// <para>
204 ///     Creates the element using the specified name.
205 /// </para>
206 /// </summary>
207 /// <param name="name">
208 /// <para>The name.</para>
209 /// </param>
210 /// <returns>
211 /// <para>The link</para>
212 /// </returns>
213 public TLink CreateElement(string name) => Create(_elementMarker, name);
214
215 /// <summary>
216 /// <para>
217 ///     Creates the text element using the specified content.
218 /// </para>
219 /// </summary>
220 /// <param name="content">
221 /// <para>The content.</para>
222 /// </param>

```

```

217     /// <para></para>
218     /// </param>
219     /// <returns>
220     /// <para>The link</para>
221     /// <para></para>
222     /// </returns>
223     public TLink CreateTextElement(string content) => Create(_textElementMarker, content);
224     /// <summary>
225     /// <para>
226     /// Creates the marker.
227     /// </para>
228     /// <para></para>
229     /// </summary>
230     /// <param name="marker">
231     /// <para>The marker.</para>
232     /// <para></para>
233     /// </param>
234     /// <param name="content">
235     /// <para>The content.</para>
236     /// <para></para>
237     /// </param>
238     /// <returns>
239     /// <para>The link</para>
240     /// <para></para>
241     /// </returns>
242     private TLink Create(TLink marker, string content) => _links.GetOrCreate(marker,
    ↪ _stringToUnicodeSequenceConverter.Convert(content));
243     /// <summary>
244     /// <para>
245     /// Attaches the element to parent using the specified element to attach.
246     /// </para>
247     /// <para></para>
248     /// </summary>
249     /// <param name="elementToAttach">
250     /// <para>The element to attach.</para>
251     /// <para></para>
252     /// </param>
253     /// <param name="parent">
254     /// <para>The parent.</para>
255     /// <para></para>
256     /// </param>
257     public void AttachElementToParent(TLink elementToAttach, TLink parent) =>
    ↪ _links.GetOrCreate(parent, elementToAttach);
258     /// <summary>
259     /// <para>
260     /// Gets the document using the specified name.
261     /// </para>
262     /// <para></para>
263     /// </summary>
264     /// <param name="name">
265     /// <para>The name.</para>
266     /// <para></para>
267     /// </param>
268     /// <returns>
269     /// <para>The link</para>
270     /// <para></para>
271     /// </returns>
272     public TLink GetDocument(string name) => Get(_documentMarker, name);
273     /// <summary>
274     /// <para>
275     /// Gets the text element using the specified content.
276     /// </para>
277     /// <para></para>
278     /// </summary>
279     /// <param name="content">
280     /// <para>The content.</para>
281     /// <para></para>
282     /// </param>
283     /// <returns>
284     /// <para>The link</para>
285     /// <para></para>
286     /// </returns>
287     public TLink GetTextElement(string content) => Get(_textElementMarker, content);
288     /// <summary>
289     /// <para>
290     /// Gets the element using the specified name.

```

```

292     /// </para>
293     /// <para></para>
294     /// </summary>
295     /// <param name="name">
296     /// <para>The name.</para>
297     /// <para></para>
298     /// </param>
299     /// <returns>
300     /// <para>The link</para>
301     /// <para></para>
302     /// </returns>
303     public TLink GetElement(string name) => Get(_elementMarker, name);
304     /// <summary>
305     /// <para>
306     /// Gets the marker.
307     /// </para>
308     /// <para></para>
309     /// </summary>
310     /// <param name="marker">
311     /// <para>The marker.</para>
312     /// <para></para>
313     /// </param>
314     /// <param name="content">
315     /// <para>The content.</para>
316     /// <para></para>
317     /// </param>
318     /// <returns>
319     /// <para>The link</para>
320     /// <para></para>
321     /// </returns>
322     private TLink Get(TLink marker, string content) => _links.SearchOrDefault(marker,
        ↪ _stringToUnicodeSequenceConverter.Convert(content));
323     /// <summary>
324     /// <para>
325     /// Gets the children using the specified parent.
326     /// </para>
327     /// <para></para>
328     /// </summary>
329     /// <param name="parent">
330     /// <para>The parent.</para>
331     /// <para></para>
332     /// </param>
333     /// <returns>
334     /// <para>The childrens.</para>
335     /// <para></para>
336     /// </returns>
337     public IList<TLink> GetChildren(TLink parent) {
338         var childrens = new List<TLink>();
339         _links.Each((link) => {
340             childrens.Add(_links.GetTarget(link));
341             return this._links.Constants.Continue;
342         }, new Link<TLink>(_links.Constants.Any, parent, _links.Constants.Any));
343         return childrens;
344     }
345 }
346 }

```

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<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
70 /// <summary>
71 /// <para>
72 /// Counts the reader.
73 /// </para>
74 /// <para></para>
75 /// </summary>
76 /// <param name="reader">
77 /// <para>The reader.</para>
78 /// <para></para>
79 /// </param>
80 /// <param name="elementNameToCount">
81 /// <para>The element name to count.</para>
82 /// <para></para>
83 /// </param>
84 /// <param name="token">
85 /// <para>The token.</para>
86 /// <para></para>
87 /// </param>
88 /// <param name="context">
89 /// <para>The context.</para>
90 /// <para></para>
91 /// </param>
92 private void Count(XmlReader reader, string elementNameToCount, CancellationToken token,
93     ↳ XmlElementContext context)
94 {
95     var rootContext = (RootElementContext)context;
96     var parentContexts = new Stack<XmlElementContext>();
97     var elements = new Stack<string>(); // Path
98     // TODO: If path was loaded previously, skip it.
99     while (reader.Read())
100     {
101         if (token.IsCancellationRequested)
102         {
103             return;
104         }
105         switch (reader.NodeType)

```

```

103     {
104         case XmlNodeType.Element:
105             var elementName = reader.Name;
106             context.IncrementChildNameCount(elementName);
107             elementName =
108                 → $"{elementName}[{context.ChildrenNamesCounts[elementName]}]";
109             if (!reader.IsEmptyElement)
110             {
111                 elements.Push(elementName);
112                 ConsoleHelpers.Debug("{0} starting...", elements.Count <= 20 ?
113                     → ToXPath(elements) : elementName); // XPath
114                 parentContexts.Push(context);
115                 context = new XmlElementContext();
116             }
117             else
118             {
119                 ConsoleHelpers.Debug("{0} finished.", elementName);
120             }
121             break;
122         case XmlNodeType.EndElement:
123             ConsoleHelpers.Debug("{0} finished.", elements.Count <= 20 ?
124                 → ToXPath(elements) : elements.Peek()); // XPath
125             var topElement = elements.Pop();
126             // Restoring scope
127             context = parentContexts.Pop();
128             if (topElement.StartsWith(elementNameToCount))
129             {
130                 rootContext.TotalElements++;
131                 // TODO: Check for 0x00 part/symbol at 198102797 line and 13
132                 → position.
133                 //if (rootContext.TotalPages > 3490000)
134                 //    selfCancel = true;
135                 if (context.ChildrenNamesCounts[elementNameToCount] % 10000 == 0)
136                 {
137                     Console.WriteLine(topElement);
138                 }
139             }
140             break;
141         case XmlNodeType.Text:
142             ConsoleHelpers.Debug("Starting text element...");
143             var content = reader.Value;
144             rootContext.TotalContentsLength += (ulong)content.Length;
145             ConsoleHelpers.Debug($"{Content length is: {content.Length}");
146             ConsoleHelpers.Debug("Text element finished.");
147             break;
148     }
149 }
150
151 /// <summary>
152 /// <para>
153 /// Returns the x path using the specified path.
154 /// </para>
155 /// <para></para>
156 /// </summary>
157 /// <param name="path">
158 /// <para>The path.</para>
159 /// <para></para>
160 /// </param>
161 /// <returns>
162 /// <para>The string</para>
163 /// <para></para>
164 /// </returns>
165 private string ToXPath(Stack<string> path) => string.Join("/", path.Reverse());
166
167 /// <summary>
168 /// <para>
169 /// Represents the root element context.
170 /// </para>
171 /// <para></para>
172 /// </summary>
173 /// <seealso cref="XmlElementContext"/>
174 private class RootElementContext : XmlElementContext
175 {
176     /// <summary>
177     /// <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         /// <summary>
79         /// <para>
80         /// Writes the writer.
81         /// </para>
82         /// <para></para>
83         /// </summary>
84         /// <param name="writer">
85         /// <para>The writer.</para>
86         /// <para></para>
87         /// </param>
88         /// <param name="token">

```

```

89     /// <para>The token.</para>
90     /// <para></para>
91     /// </param>
92     /// <param name="context">
93     /// <para>The context.</para>
94     /// <para></para>
95     /// </param>
96     private void Write(XmlWriter writer, CancellationToken token, ElementContext context)
97     {
98         var parentContexts = new Stack<ElementContext>();
99         var elements = new Stack<string>(); // Path
100                                     // TODO: If path was loaded previously, skip it.
101         foreach(TLink lvl in _storage.GetChildren(parent: context.Parent))
102         {
103             Write(writer: writer, token: token, context: new ElementContext(lvl));
104         }
105     }
106
107     /// <summary>
108     /// <para>
109     /// Represents the element context.
110     /// </para>
111     /// <para></para>
112     /// </summary>
113     /// <seealso cref="XmlElementContext"/>
114     private class ElementContext : XmlElementContext
115     {
116         /// <summary>
117         /// <para>
118         /// The parent.
119         /// </para>
120         /// <para></para>
121         /// </summary>
122         public readonly TLink Parent;
123
124         /// <summary>
125         /// <para>
126         /// Initializes a new <see cref="ElementContext"/> instance.
127         /// </para>
128         /// <para></para>
129         /// </summary>
130         /// <param name="parent">
131         /// <para>A parent.</para>
132         /// <para></para>
133         /// </param>
134         public ElementContext(TLink parent) => Parent = parent;
135     }
136
137 }
138 }

```

1.8 ./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 /// <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 /// <summary>
77 /// <para>
78 /// Reads the reader.
79 /// </para>
80 /// <para></para>
81 /// </summary>
82 /// <param name="reader">
83 /// <para>The reader.</para>
84 /// <para></para>
85 /// </param>
86 /// <param name="token">
87 /// <para>The token.</para>
88 /// <para></para>
89 /// </param>
90 /// <param name="context">
91 /// <para>The context.</para>
92 /// <para></para>
93 /// </param>
94 private void Read(XmlReader reader, CancellationToken token, ElementContext context)
95 {
96     var parentContexts = new Stack<ElementContext>();
97     var elements = new Stack<string>(); // Path
98     // TODO: If path was loaded previously, skip it.
99     while (reader.Read())
100     {
101         if (token.IsCancellationRequested)
102         {
103             return;
104         }
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);
119                     parentContexts.Push(context);
120                     _storage.AttachElementToParent(elementToAttach: element, parent:
121                         ↪ context.Parent);
122                     context = new ElementContext(element);
123                 }
124                 else
125                 {
126                     ConsoleHelpers.Debug("{0} finished.", elementName);
127                 }
128                 break;
129             case XmlNodeType.EndElement:
130                 ConsoleHelpers.Debug("{0} finished.", elements.Count <= 20 ?
131                     ↪ ToXPath(elements) : elements.Peek()); // XPath
132                 elements.Pop();
133                 // Restoring scope
134                 context = parentContexts.Pop();
135                 if (elements.Count == 1)
136                 {
137                     if (context.TotalChildren % 10 == 0)
138                         Console.WriteLine(context.TotalChildren);
139                 }
140                 break;
141             case XmlNodeType.Text:
142                 ConsoleHelpers.Debug("Starting text element...");
143                 var content = reader.Value;
144                 ConsoleHelpers.Debug("Content: {0}{1}", content.Truncate(50),
145                     ↪ content.Length >= 50 ? "... " : "");
146                 var textElement = _storage.CreateTextElement(content: content);
147                 _storage.AttachElementToParent(textElement, context.Parent);
148                 ConsoleHelpers.Debug("Text element finished.");
149                 break;
150         }
151     }
152 }
153
154 /// <summary>
155 /// <para>
156 /// Returns the x path using the specified path.
157 /// </para>
158 /// <para></para>
159 /// </summary>
160 /// <param name="path">
161 /// <para>The path.</para>
162 /// <para></para>
163 /// </param>
164 /// <returns>
165 /// <para>The string</para>
166 /// <para></para>
167 /// </returns>
168 private string ToXPath(Stack<string> path) => string.Join("/", path.Reverse());
169
170 /// <summary>
171 /// <para>
172 /// Represents the element context.
173 /// </para>
174 /// <para></para>
175 /// </summary>
176 /// <seealso cref="XmlElementContext"/>
177 private class ElementContext : XmlElementContext
178 {
179     /// <summary>
180     /// <para>
181     /// The parent.
182     /// </para>
183     /// <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 }

```

1.9 ./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);

```



```

60         importer.Import(file, cancellation.Token).Wait();
61     }
62 }
63 }
64 }
65 }
66 }

```

1.10 ./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     /// <para></para>
103     /// </summary>
104     /// <param name="links">
105     /// <para>The links.</para>
106     /// <para></para>
107     /// </param>
108     private void InitConstants(ILinks<TLink> links)
109     {
110         var markerIndex = _one;
111         var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
112         _unicodeSymbolMarker = links.GetOrCreate(meaningRoot,
113             ↪ Arithmetic.Increment(markerIndex));
114         _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
115         _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
116         _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
117         _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
118     }
119
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     public void AttachElementToParent(TLink elementToAttach, TLink parent)
135     {
136     }
137
138     /// <summary>
139     /// <para>
140     /// Returns the elements using the specified string.
141     /// </para>
142     /// <para></para>
143     /// </summary>
144     /// <param name="@string">
145     /// <para>The string.</para>
146     /// <para></para>
147     /// </param>
148     /// </returns>

```

```

146 /// <para>The elements.</para>
147 /// <para></para>
148 /// </returns>
149 public IList<TLink> ToElements(string @string)
150 {
151     var elements = new TLink[@string.Length];
152     for (int i = 0; i < @string.Length; i++)
153     {
154         elements[i] = _charToUnicodeSymbolConverter.Convert(@string[i]);
155     }
156     return elements;
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, 5
./csharp/Platform.Data.Doublets.Xml/IXmlStorage.cs, 6
./csharp/Platform.Data.Doublets.Xml/XmlElementContext.cs, 7
./csharp/Platform.Data.Doublets.Xml/XmlElementCounter.cs, 8
./csharp/Platform.Data.Doublets.Xml/XmlElementCounterCLI.cs, 11
./csharp/Platform.Data.Doublets.Xml/XmlExporter.cs, 11
./csharp/Platform.Data.Doublets.Xml/XmlImporter.cs, 13
./csharp/Platform.Data.Doublets.Xml/XmlImporterCLI.cs, 16
./csharp/Platform.Data.Doublets.Xml/XmlIndexer.cs, 17