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

```
using System.Collections.Generic;
using Platform.Numbers;
using Platform.Data.Numbers.Raw;
using Platform.Data.Doublets;
using Platform.Data.Doublets.Sequences.Converters;
using Platform.Data.Doublets.Sequences.Frequencies.Cache;
using Platform.Data.Doublets.Sequences.Indexes;
using Platform.Data.Doublets.Unicode;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Doublets.Xml
{
    public class DefaultXmlStorage<TLink> : IXmlStorage<TLink>
    {
        private static readonly TLink _zero = default;
        private static readonly TLink _one = Arithmetic.Increment(_zero);

        private readonly StringToUnicodeSequenceConverter<TLink>
        ↪  _stringToUnicodeSequenceConverter;
        private readonly ILinks<TLink> _links;
        private TLink _unicodeSymbolMarker;
        private TLink _unicodeSequenceMarker;
        private TLink _elementMarker;
        private TLink _textElementMarker;
        private TLink _documentMarker;

        private class Unindex : ISequenceIndex<TLink>
        {
            public bool Add(IList<TLink> sequence) => true;
            public bool MightContain(IList<TLink> sequence) => true;
        }

        public DefaultXmlStorage(ILinks<TLink> links, bool indexSequenceBeforeCreation,
        ↪  LinkFrequenciesCache<TLink> frequenciesCache)
        {
            var linkToItsFrequencyNumberConverter = new
            ↪  FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<TLink>(frequenciesCache);
            var sequenceToItsLocalElementLevelsConverter = new
            ↪  SequenceToItsLocalElementLevelsConverter<TLink>(links,
            ↪  linkToItsFrequencyNumberConverter);
            var optimalVariantConverter = new OptimalVariantConverter<TLink>(links,
            ↪  sequenceToItsLocalElementLevelsConverter);
            InitConstants(links);
            var charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<TLink>(links,
            ↪  new AddressToRawNumberConverter<TLink>(), _unicodeSymbolMarker);
            var index = indexSequenceBeforeCreation ? new
            ↪  CachedFrequencyIncrementingSequenceIndex<TLink>(frequenciesCache) :
            ↪  (ISequenceIndex<TLink>)new Unindex();
            _stringToUnicodeSequenceConverter = new
            ↪  StringToUnicodeSequenceConverter<TLink>(links, charToUnicodeSymbolConverter,
            ↪  index, optimalVariantConverter, _unicodeSequenceMarker);
            _links = links;
        }

        private void InitConstants(ILinks<TLink> links)
        {
            var markerIndex = _one;
            var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
            _unicodeSymbolMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
            ↪  markerIndex));
            _unicodeSequenceMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
            ↪  markerIndex));
            _elementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
            ↪  markerIndex));
            _textElementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
            ↪  markerIndex));
            _documentMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
            ↪  markerIndex));
        }

        public TLink CreateDocument(string name) => Create(_documentMarker, name);

        public TLink CreateElement(string name) => Create(_elementMarker, name);

        public TLink CreateTextElement(string content) => Create(_textElementMarker, content);

```

```
62          private TLink Create(TLink marker, string content)
63          {
64              var contentSequence = _stringToUnicodeSequenceConverter.Convert(content);
65              return _links.GetOrCreate(marker, contentSequence);
66          }
67
68          public void AttachElementToParent(TLink elementToAttach, TLink parent) =>
            ↪  _links.GetOrCreate(parent, elementToAttach);
69      }
70  }
```

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       public interface ICommandLineInterface
6       {
7           void Run(params string[] args);
8       }
9   }
```

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   namespace Platform.Data.Doublets.Xml
4   {
5       public interface IXmlStorage<TLink>
6       {
7           TLink CreateDocument(string name);
8           TLink CreateElement(string name);
9           TLink CreateTextElement(string content);
10          void AttachElementToParent(TLink elementToAttach, TLink parent);
11      }
12  }
```

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       internal class XmlElementContext
8       {
9           public readonly Dictionary<string, int> ChildrenNamesCounts;
10          public int TotalChildren;
11
12          public XmlElementContext() => ChildrenNamesCounts = new Dictionary<string, int>();
13
14          public void IncrementChildNameCount(string name)
15          {
16              if (ChildrenNamesCounts.TryGetValue(name, out int count))
17              {
18                  ChildrenNamesCounts[name] = count + 1;
19              }
20              else
21              {
22                  ChildrenNamesCounts[name] = 0;
23              }
24              TotalChildren++;
25          }
26      }
27  }
```

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      public class XmlElementCounter
15      {
16          public XmlElementCounter() { }
17
18          public Task Count(string file, string elementName, CancellationToken token)
19          {
20              return Task.Factory.StartNew(() =>
21              {
22                  try
23                  {
24                      var context = new RootElementContext();
25                      using (var reader = XmlReader.Create(file))
26                      {
27                          Count(reader, elementName, token, context);
28                      }
29                      Console.WriteLine($"Total elements with specified name:
                        ↪  {context.TotalElements}, total content length:
                        ↪  {context.TotalContentsLength}.");
30                  }
31                  catch (Exception ex)
32                  {
33                      Console.WriteLine(ex.ToStringWithAllInnerExceptions());
34                  }
35              }, token);
36          }
37
38          private void Count(XmlReader reader, string elementNameToCount, CancellationToken token,
            ↪  XmlElementContext context)
39          {
40              var rootContext = (RootElementContext)context;
41              var parentContexts = new Stack<XmlElementContext>();
42              var elements = new Stack<string>(); // Path
43              // TODO: If path was loaded previously, skip it.
44              while (reader.Read())
45              {
46                  if (token.IsCancellationRequested)
47                  {
48                      return;
49                  }
50                  switch (reader.NodeType)
51                  {
52                      case XmlNodeType.Element:
53                          var elementName = reader.Name;
54                          context.IncrementChildNameCount(elementName);
55                          elementName =
                            ↪  $"{elementName}[{context.ChildrenNamesCounts[elementName]}]";
56                          if (!reader.IsEmptyElement)
57                          {
58                              elements.Push(elementName);
59                              ConsoleHelpers.Debug("{0} starting...", elements.Count <= 20 ?
                                ↪  ToXPath(elements) : elementName); // XPath
60                              parentContexts.Push(context);
61                              context = new XmlElementContext();
62                          }
63                          else
64                          {
65                              ConsoleHelpers.Debug("{0} finished.", elementName);
66                          }
67                          break;
68
69                      case XmlNodeType.EndElement:
70                          ConsoleHelpers.Debug("{0} finished.", elements.Count <= 20 ?
                            ↪  ToXPath(elements) : elements.Peek()); // XPath
71                          var topElement = elements.Pop();
72                          // Restoring scope
73                          context = parentContexts.Pop();
74                          if (topElement.StartsWith(elementNameToCount))
75                          {
76                              rootContext.TotalElements++;
77                              // TODO: Check for 0x00 part/symbol at 198102797 line and 13
                                ↪  position.
78                              //if (rootContext.TotalPages > 3490000)
79                              //    selfCancel = true;
80                              if (context.ChildrenNamesCounts[elementNameToCount] % 10000 == 0)
81                              {
82                                  Console.WriteLine(topElement);
83                              }
84                          }
```

```
                        break;

                    case XmlNodeType.Text:
                        ConsoleHelpers.Debug("Starting text element...");
                        var content = reader.Value;
                        rootContext.TotalContentsLength += (ulong)content.Length;
                        ConsoleHelpers.Debug($"Content length is: {content.Length}");
                        ConsoleHelpers.Debug("Text element finished.");
                        break;
                }
            }
        }

        private string ToXPath(Stack<string> path) => string.Join("/", path.Reverse());

        private class RootElementContext : XmlElementContext
        {
            public ulong TotalElements;
            public ulong TotalContentsLength;
        }
    }
}
```

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

```
using System;
using System.IO;
using Platform.IO;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Doublets.Xml
{
    public class XmlElementCounterCLI : ICommandLineInterface
    {
        public void Run(params string[] args)
        {
            var file = ConsoleHelpers.GetOrReadArgument(0, "Xml file", args);
            var elementName = ConsoleHelpers.GetOrReadArgument(1, "Element name to count", args);
            if (!File.Exists(file))
            {
                Console.WriteLine("Entered xml file does not exists.");
            }
            else if (string.IsNullOrEmpty(elementName))
            {
                Console.WriteLine("Entered element name is empty.");
            }
            else
            {
                using (var cancellation = new ConsoleCancellation())
                {
                    Console.WriteLine("Press CTRL+C to stop.");
                    new XmlElementCounter().Count(file, elementName, cancellation.Token).Wait();
                }
            }
        }
    }
}
```

1.7 ./csharp/Platform.Data.Doublets.Xml/XmlImporter.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
using System.Xml;
using Platform.Exceptions;
using Platform.Collections;
using Platform.IO;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.Data.Doublets.Xml {
    public class XmlImporter<TLink>
    {
        private readonly IXmlStorage<TLink> _storage;

        public XmlImporter(IXmlStorage<TLink> storage) => _storage = storage;

        public Task Import(string file, CancellationToken token)
        {
```

```
22              return Task.Factory.StartNew(() =>
23              {
24                  try
25                  {
26                      var document = _storage.CreateDocument(file);
27
28                      using (var reader = XmlReader.Create(file))
29                      {
30                          Read(reader, token, new ElementContext(document));
31                      }
32                  }
33                  catch (Exception ex)
34                  {
35                      Console.WriteLine(ex.ToStringWithAllInnerExceptions());
36                  }
37
38              }, token);
39          }
40
41          private void Read(XmlReader reader, CancellationToken token, ElementContext context)
42          {
43              var parentContexts = new Stack<ElementContext>();
44              var elements = new Stack<string>(); // Path
45              // TODO: If path was loaded previously, skip it.
46              while (reader.Read())
47              {
48                  if (token.IsCancellationRequested)
49                  {
50                      return;
51                  }
52                  switch (reader.NodeType)
53                  {
54                      case XmlNodeType.Element:
55                          var elementName = reader.Name;
56                          context.IncrementChildNameCount(elementName);
57                          elementName =
                            ↪  $"{elementName}[{context.ChildrenNamesCounts[elementName]}]";
58                          if (!reader.IsEmptyElement)
59                          {
60                              elements.Push(elementName);
61                              ConsoleHelpers.Debug("{0} starting...", elements.Count <= 20 ?
                                ↪  ToXPath(elements) : elementName); // XPath
62                              var element = _storage.CreateElement(name: elementName);
63                              parentContexts.Push(context);
64                              _storage.AttachElementToParent(elementToAttach: element, parent:
                                ↪  context.Parent);
65                              context = new ElementContext(element);
66                          }
67                          else
68                          {
69                              ConsoleHelpers.Debug("{0} finished.", elementName);
70                          }
71                          break;
72                      case XmlNodeType.EndElement:
73                          ConsoleHelpers.Debug("{0} finished.", elements.Count <= 20 ?
                            ↪  ToXPath(elements) : elements.Peek()); // XPath
74                          elements.Pop();
75                          // Restoring scope
76                          context = parentContexts.Pop();
77                          if (elements.Count == 1)
78                          {
79                              if (context.TotalChildren % 10 == 0)
80                                  Console.WriteLine(context.TotalChildren);
81                          }
82                          break;
83                      case XmlNodeType.Text:
84                          ConsoleHelpers.Debug("Starting text element...");
85                          var content = reader.Value;
86                          ConsoleHelpers.Debug("Content: {0}{1}", content.Truncate(50),
                            ↪  content.Length >= 50 ? "..." : "");
87                          var textElement = _storage.CreateTextElement(content: content);
88                          _storage.AttachElementToParent(textElement, context.Parent);
89                          ConsoleHelpers.Debug("Text element finished.");
90                          break;
91                  }
92              }
93          }
94
```

```
95          private string ToXPath(Stack<string> path) => string.Join("/", path.Reverse());
96
97          private class ElementContext : XmlElementContext
98          {
99              public readonly TLink Parent;
100
101             public ElementContext(TLink parent)
102             {
103                 Parent = parent;
104             }
105         }
106     }
107 }
```

1.8 ./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     public class XmlImporterCLI : ICommandLineInterface
11     {
12         public void Run(params string[] args)
13         {
14             var linksFile = ConsoleHelpers.GetOrReadArgument(0, "Links file", args);
15             var file = ConsoleHelpers.GetOrReadArgument(1, "Xml file", args);
16
17             if (!File.Exists(file))
18             {
19                 Console.WriteLine("Entered xml file does not exists.");
20             }
21             else
22             {
23                 //const long gb32 = 34359738368;
24
25                 using (var cancellation = new ConsoleCancellation())
26                 using (var memoryAdapter = new UnitedMemoryLinks<uint>(linksFile))
27                 //using (var memoryAdapter = new UInt64UnitedMemoryLinks(linksFile, gb32))
28                 //using (var links = new UInt64Links(memoryAdapter))
29                 {
30                     Console.WriteLine("Press CTRL+C to stop.");
31                     var links =
                        ↪ memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
32                     var indexer = new XmlIndexer<uint>(links);
33                     var indexingImporter = new XmlImporter<uint>(indexer);
34                     indexingImporter.Import(file, cancellation.Token).Wait();
35                     if (cancellation.NotRequested)
36                     {
37                         var cache = indexer.Cache;
38                         //var counter = new TotalSequenceSymbolFrequencyCounter<uint>(links);
39                         //var cache = new LinkFrequenciesCache<uint>(links, counter);
40                         Console.WriteLine("Frequencies cache ready.");
41                         var storage = new DefaultXmlStorage<uint>(links, false, cache);
42                         var importer = new XmlImporter<uint>(storage);
43                         importer.Import(file, cancellation.Token).Wait();
44                     }
45                 }
46             }
47         }
48     }
49 }
```

1.9 ./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 {
```

```
    public class XmlIndexer<TLink> : IXmlStorage<TLink>
    {
        private static readonly TLink _zero = default;
        private static readonly TLink _one = Arithmetic.Increment(_zero);

        private readonly CachedFrequencyIncrementingSequenceIndex<TLink> _index;
        private readonly CharToUnicodeSymbolConverter<TLink> _charToUnicodeSymbolConverter;
        private TLink _unicodeSymbolMarker;
        private readonly TLink _nullConstant;

        public LinkFrequenciesCache<TLink> Cache { get; }

        public XmlIndexer(ILinks<TLink> links)
        {
            _nullConstant = links.Constants.Null;
            var totalSequenceSymbolFrequencyCounter = new
            ↪  TotalSequenceSymbolFrequencyCounter<TLink>(links);
            Cache = new LinkFrequenciesCache<TLink>(links, totalSequenceSymbolFrequencyCounter);
            _index = new CachedFrequencyIncrementingSequenceIndex<TLink>(Cache);
            var addressToRawNumberConverter = new AddressToRawNumberConverter<TLink>();
            InitConstants(links);
            _charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<TLink>(links,
            ↪  addressToRawNumberConverter, _unicodeSymbolMarker);
        }

        private void InitConstants(ILinks<TLink> links)
        {
            var markerIndex = _one;
            var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
            _unicodeSymbolMarker = links.GetOrCreate(meaningRoot,
            ↪  Arithmetic.Increment(markerIndex));
            _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
            _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
            _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
            _ = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
        }

        public void AttachElementToParent(TLink elementToAttach, TLink parent)
        {
        }

        public IList<TLink> ToElements(string @string)
        {
            var elements = new TLink[@string.Length];
            for (int i = 0; i < @string.Length; i++)
            {
                elements[i] = _charToUnicodeSymbolConverter.Convert(@string[i]);
            }
            return elements;
        }

        public TLink CreateDocument(string name)
        {
            _index.Add(ToElements(name));
            return _nullConstant;
        }

        public TLink CreateElement(string name)
        {
            _index.Add(ToElements(name));
            return _nullConstant;
        }

        public TLink CreateTextElement(string content)
        {
            _index.Add(ToElements(content));
            return _nullConstant;
        }
    }
}
```

Index