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.Frequencies.Counters;
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
{
    /// <summary>
    /// <para>
    /// Represents the default xml storage.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="IXmlStorage{TLink}"/>
    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>
        {
            /// <summary>
            /// <para>
            /// Determines whether this instance add.
            /// </para>
            /// <para></para>
            /// </summary>
            /// <param name="sequence">
            /// <para>The sequence.</para>
            /// <para></para>
            /// </param>
            /// <returns>
            /// <para>The bool</para>
            /// <para></para>
            /// </returns>
            public bool Add(IList<TLink> sequence) => true;

            /// <summary>
            /// <para>
            /// Determines whether this instance might contain.
            /// </para>
            /// <para></para>
            /// </summary>
            /// <param name="sequence">
            /// <para>The sequence.</para>
            /// <para></para>
            /// </param>
            /// <returns>
            /// <para>The bool</para>
            /// <para></para>
            /// </returns>
            public bool MightContain(IList<TLink> sequence) => true;
        }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="DefaultXmlStorage"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <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,
            ↪ LinkFrequenciesCache<TLink> frequenciesCache)
88          {
89              var linkToItsFrequencyNumberConverter = new
                ↪ FrequenciesCacheBasedLinkToItsFrequencyNumberConverter<TLink>(frequenciesCache);
90              var sequenceToItsLocalElementLevelsConverter = new
                ↪ SequenceToItsLocalElementLevelsConverter<TLink>(links,
                ↪ linkToItsFrequencyNumberConverter);
91              var optimalVariantConverter = new OptimalVariantConverter<TLink>(links,
                ↪ sequenceToItsLocalElementLevelsConverter);
92              InitConstants(links);
93              var charToUnicodeSymbolConverter = new CharToUnicodeSymbolConverter<TLink>(links,
                ↪ new AddressToRawNumberConverter<TLink>(), _unicodeSymbolMarker);
94              var index = indexSequenceBeforeCreation ? new
                ↪ CachedFrequencyIncrementingSequenceIndex<TLink>(frequenciesCache) :
                ↪ (ISequenceIndex<TLink>)new Unindex();
95              _stringToUnicodeSequenceConverter = new
                ↪ StringToUnicodeSequenceConverter<TLink>(links, charToUnicodeSymbolConverter,
                ↪ index, optimalVariantConverter, _unicodeSequenceMarker);
96              _links = links;
97          }
98
99          public DefaultXmlStorage(ILinks<TLink> links, bool indexSequenceBeforeCreation = false) :
100             this(links, indexSequenceBeforeCreation,
101                 new LinkFrequenciesCache<TLink>(links,
102                     new TotalSequenceSymbolFrequencyCounter<TLink>(links))) { }
103
104         private void InitConstants(ILinks<TLink> links)
105         {
106             var markerIndex = _one;
107             var meaningRoot = links.GetOrCreate(markerIndex, markerIndex);
108             _unicodeSymbolMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
109             _unicodeSequenceMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
110             _elementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
111             _textElementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
112             _documentMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
113         }
114
115         /// <summary>
116         /// <para>
117         /// Creates the document using the specified name.
118         /// </para>
119         /// <para></para>
120         /// </summary>
121         /// <param name="name">
122         /// <para>The name.</para>
123         /// <para></para>
124         /// </param>
125         /// <returns>
126         /// <para>The link</para>
127         /// <para></para>
128         /// </returns>
129         public TLink CreateDocument(string name) => Create(_documentMarker, name);
130
131         /// <summary>
132         /// <para>
133         /// Creates the element using the specified name.
134         /// </para>
135         /// <para></para>
136         /// </summary>
137         /// <param name="name">
138         /// <para>The name.</para>
```

```
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink CreateElement(string name) => Create(_elementMarker, name);

        /// <summary>
        /// <para>
        /// Creates the text element using the specified content.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="content">
        /// <para>The content.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink CreateTextElement(string content) => Create(_textElementMarker, content);

        private TLink Create(TLink marker, string content) => _links.GetOrCreate(marker,
        ↪  _stringToUnicodeSequenceConverter.Convert(content));

        /// <summary>
        /// <para>
        /// Attaches the element to parent using the specified element to attach.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="elementToAttach">
        /// <para>The element to attach.</para>
        /// <para></para>
        /// </param>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        public void AttachElementToParent(TLink elementToAttach, TLink parent) =>
        ↪  _links.GetOrCreate(parent, elementToAttach);

        /// <summary>
        /// <para>
        /// Gets the document using the specified name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="name">
        /// <para>The name.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetDocument(string name) => Get(_documentMarker, name);

        /// <summary>
        /// <para>
        /// Gets the text element using the specified content.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="content">
        /// <para>The content.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetTextElement(string content) => Get(_textElementMarker, content);
        /// <summary>
        /// <para>
        /// Gets the element using the specified name.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="name">
        /// <para>The name.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetElement(string name) => Get(_elementMarker, name);

        private TLink Get(TLink marker, string content) => _links.SearchOrDefault(marker,
        ↪  _stringToUnicodeSequenceConverter.Convert(content));

        /// <summary>
        /// <para>
        /// Gets the children using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The childrens.</para>
        /// <para></para>
        /// </returns>
        public IList<TLink> GetChildren(TLink parent) {
            List<TLink> childrens = new List<TLink>();
            _links.Each((link) => {
                childrens.Add(_links.GetTarget(link));
                return this._links.Constants.Continue;
            }, new Link<TLink>(_links.Constants.Any, parent, _links.Constants.Any));
            return childrens;
        }
    }
}
```

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

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

namespace Platform.Data.Doublets.Xml
{
    /// <summary>
    /// <para>
    /// Defines the command line interface.
    /// </para>
    /// <para></para>
    /// </summary>
    public interface ICommandLineInterface
    {
        /// <summary>
        /// <para>
        /// Runs the args.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="args">
        /// <para>The args.</para>
        /// <para></para>
        /// </param>
        void Run(params string[] args);
    }
}
```

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

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

using System.Collections.Generic;

namespace Platform.Data.Doublets.Xml
{
    /// <summary>
    /// <para>
    /// 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>
```

```
        /// </returns>
        TLink GetElement(string name);
        /// <summary>
        /// <para>
        /// Gets the text element using the specified content.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="content">
        /// <para>The content.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink GetTextElement(string content);
        /// <summary>
        /// <para>
        /// Gets the children using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>A list of i list t link</para>
        /// <para></para>
        /// </returns>
        IList<TLink> GetChildren(TLink parent);
        /// <summary>
        /// <para>
        /// Attaches the element to parent using the specified element to attach.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="elementToAttach">
        /// <para>The element to attach.</para>
        /// <para></para>
        /// </param>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        void AttachElementToParent(TLink elementToAttach, TLink parent);
    }
}
```

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

```
using System.Collections.Generic;

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

namespace Platform.Data.Doublets.Xml
{
    /// <summary>
    /// <para>
    /// Represents the xml element context.
    /// </para>
    /// <para></para>
    /// </summary>
    internal class XmlElementContext
    {
        public readonly Dictionary<string, int> ChildrenNamesCounts;
        public int TotalChildren;

        public XmlElementContext() => ChildrenNamesCounts = new Dictionary<string, int>();

        public void IncrementChildNameCount(string name)
        {
            if (ChildrenNamesCounts.TryGetValue(name, out int count))
            {
                ChildrenNamesCounts[name] = count + 1;
            }
            else
            {
                ChildrenNamesCounts[name] = 0;
```

```
            }
            TotalChildren++;
        }
    }
}
```

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

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

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

namespace Platform.Data.Doublets.Xml
{
    /// <summary>
    /// <para>
    /// Represents the xml element counter.
    /// </para>
    /// <para></para>
    /// </summary>
    public class XmlElementCounter
    {
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="XmlElementCounter"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        public XmlElementCounter() { }

        /// <summary>
        /// <para>
        /// Counts the file.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="file">
        /// <para>The file.</para>
        /// <para></para>
        /// </param>
        /// <param name="elementName">
        /// <para>The element name.</para>
        /// <para></para>
        /// </param>
        /// <param name="token">
        /// <para>The token.</para>
        /// <para></para>
        /// </param>
        public Task Count(string file, string elementName, CancellationToken token)
        {
            return Task.Factory.StartNew(() =>
            {
                try
                {
                    var context = new RootElementContext();
                    using (var reader = XmlReader.Create(file))
                    {
                        Count(reader, elementName, token, context);
                    }
                    Console.WriteLine($"Total elements with specified name:
                    ↪  {context.TotalElements}, total content length:
                    ↪  {context.TotalContentsLength}.");
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.ToStringWithAllInnerExceptions());
                }
            }, token);
        }

        private void Count(XmlReader reader, string elementNameToCount, CancellationToken token,
        ↪  XmlElementContext context)
        {
```

```
            var rootContext = (RootElementContext)context;
            var parentContexts = new Stack<XmlElementContext>();
            var elements = new Stack<string>(); // Path
            // TODO: If path was loaded previously, skip it.
            while (reader.Read())
            {
                if (token.IsCancellationRequested)
                {
                    return;
                }
                switch (reader.NodeType)
                {
                    case XmlNodeType.Element:
                        var elementName = reader.Name;
                        context.IncrementChildNameCount(elementName);
                        elementName =
                        ↪  $"{elementName}[{context.ChildrenNamesCounts[elementName]}]";
                        if (!reader.IsEmptyElement)
                        {
                            elements.Push(elementName);
                            ConsoleHelpers.Debug("{0} starting...", elements.Count <= 20 ?
                            ↪  ToXPath(elements) : elementName); // XPath
                            parentContexts.Push(context);
                            context = new XmlElementContext();
                        }
                        else
                        {
                            ConsoleHelpers.Debug("{0} finished.", elementName);
                        }
                        break;

                    case XmlNodeType.EndElement:
                        ConsoleHelpers.Debug("{0} finished.", elements.Count <= 20 ?
                        ↪  ToXPath(elements) : elements.Peek()); // XPath
                        var topElement = elements.Pop();
                        // Restoring scope
                        context = parentContexts.Pop();
                        if (topElement.StartsWith(elementNameToCount))
                        {
                            rootContext.TotalElements++;
                            // TODO: Check for 0x00 part/symbol at 198102797 line and 13
                            ↪  position.
                            //if (rootContext.TotalPages > 3490000)
                            //    selfCancel = true;
                            if (context.ChildrenNamesCounts[elementNameToCount] % 10000 == 0)
                            {
                                Console.WriteLine(topElement);
                            }
                        }
                        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
        {
            /// <summary>
            /// <para>
            /// The total elements.
            /// </para>
            /// <para></para>
            /// </summary>
            public ulong TotalElements;
            /// <summary>
            /// <para>
            /// The total contents length.
            /// </para>
            /// <para></para>
            /// </summary>
```

```
            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
{
    /// <summary>
    /// <para>
    /// Represents the xml element counter cli.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ICommandLineInterface"/>
    public class XmlElementCounterCLI : ICommandLineInterface
    {
        /// <summary>
        /// <para>
        /// Runs the args.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="args">
        /// <para>The args.</para>
        /// <para></para>
        /// </param>
        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/XmlExporter.cs

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

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

namespace Platform.Data.Doublets.Xml
{
    /// <summary>
    /// <para>
    /// Represents the xml exporter.
    /// </para>
    /// <para></para>
    /// </summary>
    public class XmlExporter<TLink>
    {
        private readonly IXmlStorage<TLink> _storage;
```

```

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="XmlExporter"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storage">
        /// <para>A storage.</para>
        /// <para></para>
        /// </param>
        public XmlExporter(IXmlStorage<TLink> storage) => _storage = storage;

        /// <summary>
        /// <para>
        /// Exports the document name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="documentName">
        /// <para>The document name.</para>
        /// <para></para>
        /// </param>
        /// <param name="fileName">
        /// <para>The file name.</para>
        /// <para></para>
        /// </param>
        /// <param name="token">
        /// <para>The token.</para>
        /// <para></para>
        /// </param>
        public Task Export(string documentName, string fileName, CancellationToken token)
        {
            return Task.Factory.StartNew(() =>
            {
                try
                {
                    var document = _storage.GetDocument(documentName);
                    using (var writer = XmlWriter.Create(fileName))
                    {
                        Write(writer, token, new ElementContext(document));
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.ToStringWithAllInnerExceptions());
                }
            }, token);
        }

        private void Write(XmlWriter writer, CancellationToken token, ElementContext context)
        {
            var parentContexts = new Stack<ElementContext>();
            var elements = new Stack<string>(); // Path
                                                // TODO: If path was loaded previously, skip it.
            foreach(TLink lvl in _storage.GetChildren(parent: context.Parent))
            {
                Write(writer: writer, token: token, context: new ElementContext(lvl));
            }
        }

        private class ElementContext : XmlElementContext
        {
            public readonly TLink Parent;
            public ElementContext(TLink parent) => Parent = parent;
        }

    }
}
```

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

```
using System;
using System.IO;
using Platform.IO;
using Platform.Data.Doublets.Memory.United.Generic;

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

namespace Platform.Data.Doublets.Xml
```

```
{
    public class XmlExporterCLI : ICommandLineInterface
    {
        public void Run(params string[] args)
        {
            var linksFile = ConsoleHelpers.GetOrReadArgument(0, "Links file", args);
            var exportFile = ConsoleHelpers.GetOrReadArgument(1, "Xml file", args);

            if (File.Exists(exportFile))
            {
                Console.WriteLine("Entered xml file does already exists.");
            }
            else
            {
                using (var cancellation = new ConsoleCancellation())
                {
                    using (var memoryAdapter = new UnitedMemoryLinks<uint>(linksFile))
                    {
                        {
                            Console.WriteLine("Press CTRL+C to stop.");
                            var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesRe
                            ↪  solution();
                            if (cancellation.NotRequested)
                            {
                                var storage = new DefaultXmlStorage<uint>(links);
                                var exporter = new XmlExporter<uint>(storage);
                                exporter.Export(linksFile, exportFile,
                                ↪  cancellation.Token).Wait();
                            }
                        }
                    }
                }
            }
        }
    }
}
```

1.9 ./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 {
    /// <summary>
    /// <para>
    /// Represents the xml importer.
    /// </para>
    /// <para></para>
    /// </summary>
    public class XmlImporter<TLink>
    {
        private readonly IXmlStorage<TLink> _storage;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="XmlImporter"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storage">
        /// <para>A storage.</para>
        /// <para></para>
        /// </param>
        public XmlImporter(IXmlStorage<TLink> storage) => _storage = storage;

        /// <summary>
        /// <para>
        /// Imports the file.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
        /// <param name="file">
        /// <para>The file.</para>
        /// <para></para>
        /// </param>
        /// <param name="token">
        /// <para>The token.</para>
        /// <para></para>
        /// </param>
        public Task Import(string file, CancellationToken token)
        {
            return Task.Factory.StartNew(() =>
            {
                try
                {
                    var document = _storage.CreateDocument(file);

                    using (var reader = XmlReader.Create(file))
                    {
                        Read(reader, token, new ElementContext(document));
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.ToStringWithAllInnerExceptions());
                }

            }, token);
        }

        private void Read(XmlReader reader, CancellationToken token, ElementContext context)
        {
            var parentContexts = new Stack<ElementContext>();
            var elements = new Stack<string>(); // Path
            // TODO: If path was loaded previously, skip it.
            while (reader.Read())
            {
                if (token.IsCancellationRequested)
                {
                    return;
                }
                switch (reader.NodeType)
                {
                    case XmlNodeType.Element:
                        var elementName = reader.Name;
                        context.IncrementChildNameCount(elementName);
                        elementName =
                        ↪  $"{elementName}[{context.ChildrenNamesCounts[elementName]}]";
                        if (!reader.IsEmptyElement)
                        {
                            elements.Push(elementName);
                            ConsoleHelpers.Debug("{0} starting...", elements.Count <= 20 ?
                            ↪  ToXPath(elements) : elementName); // XPath
                            var element = _storage.CreateElement(name: elementName);
                            parentContexts.Push(context);
                            _storage.AttachElementToParent(elementToAttach: element, parent:
                            ↪  context.Parent);
                            context = new ElementContext(element);
                        }
                        else
                        {
                            ConsoleHelpers.Debug("{0} finished.", elementName);
                        }
                        break;
                    case XmlNodeType.EndElement:
                        ConsoleHelpers.Debug("{0} finished.", elements.Count <= 20 ?
                        ↪  ToXPath(elements) : elements.Peek()); // XPath
                        elements.Pop();
                        // Restoring scope
                        context = parentContexts.Pop();
                        if (elements.Count == 1)
                        {
                            if (context.TotalChildren % 10 == 0)
                                Console.WriteLine(context.TotalChildren);
                        }
                        break;
                    case XmlNodeType.Text:
                        ConsoleHelpers.Debug("Starting text element...");
                        var content = reader.Value;
```

```
                        ConsoleHelpers.Debug("Content: {0}{1}", content.Truncate(50),
                        ↪  content.Length >= 50 ? "..." : "");
                        var textElement = _storage.CreateTextElement(content: content);
                        _storage.AttachElementToParent(textElement, context.Parent);
                        ConsoleHelpers.Debug("Text element finished.");
                        break;
                    }
                }
            }

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


            private class ElementContext : XmlElementContext
            {
                public readonly TLink Parent;
                public ElementContext(TLink parent) => Parent = parent;
            }
        }
    }
```

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

```
using System;
using System.IO;
using Platform.IO;
using Platform.Data.Doublets.Memory.United.Generic;

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

namespace Platform.Data.Doublets.Xml
{
    /// <summary>
    /// <para>
    /// Represents the xml importer cli.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="ICommandLineInterface"/>
    public class XmlImporterCLI : ICommandLineInterface
    {
        /// <summary>
        /// <para>
        /// Runs the args.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="args">
        /// <para>The args.</para>
        /// <para></para>
        /// </param>
        public void Run(params string[] args)
        {
            var linksFile = ConsoleHelpers.GetOrReadArgument(0, "Links file", args);
            var file = ConsoleHelpers.GetOrReadArgument(1, "Xml file", args);

            if (!File.Exists(file))
            {
                Console.WriteLine("Entered xml file does not exists.");
            }
            else
            {
                //const long gb32 = 34359738368;

                using (var cancellation = new ConsoleCancellation())
                using (var memoryAdapter = new UnitedMemoryLinks<uint>(linksFile))
                //using (var memoryAdapter = new UInt64UnitedMemoryLinks(linksFile, gb32))
                //using (var links = new UInt64Links(memoryAdapter))
                {
                    Console.WriteLine("Press CTRL+C to stop.");
                    var links =
                    ↪  memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
                    var indexer = new XmlIndexer<uint>(links);
                    var indexingImporter = new XmlImporter<uint>(indexer);
                    indexingImporter.Import(file, cancellation.Token).Wait();
                    if (cancellation.NotRequested)
                    {
                        var cache = indexer.Cache;
                        //var counter = new TotalSequenceSymbolFrequencyCounter<uint>(links);
                        //var cache = new LinkFrequenciesCache<uint>(links, counter);
```

```
                        Console.WriteLine("Frequencies cache ready.");
                        var storage = new DefaultXmlStorage<uint>(links, false, cache);
                        var importer = new XmlImporter<uint>(storage);
                        importer.Import(file, cancellation.Token).Wait();
                    }
                }
            }
        }
    }
}
```

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

```
using System.Collections.Generic;
using Platform.Numbers;
using Platform.Data.Numbers.Raw;
using Platform.Data.Doublets;
using Platform.Data.Doublets.Sequences.Frequencies.Cache;
using Platform.Data.Doublets.Sequences.Frequencies.Counters;
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
{
    /// <summary>
    /// <para>
    /// Represents the xml indexer.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="IXmlStorage{TLink}"/>
    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;

        /// <summary>
        /// <para>
        /// Gets the cache value.
        /// </para>
        /// <para></para>
        /// </summary>
        public LinkFrequenciesCache<TLink> Cache { get; }

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="XmlIndexer"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="links">
        /// <para>A links.</para>
        /// <para></para>
        /// </param>
        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));
        }

        /// <summary>
        /// <para>
        /// Attaches the element to parent using the specified element to attach.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="elementToAttach">
        /// <para>The element to attach.</para>
        /// <para></para>
        /// </param>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        public void AttachElementToParent(TLink elementToAttach, TLink parent)
        {
        }

        /// <summary>
        /// <para>
        /// Returns the elements using the specified string.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@string">
        /// <para>The string.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The elements.</para>
        /// <para></para>
        /// </returns>
        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;
        }

        /// <summary>
        /// <para>
        /// Creates the document using the specified name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="name">
        /// <para>The name.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The null constant.</para>
        /// <para></para>
        /// </returns>
        public TLink CreateDocument(string name)
        {
            _index.Add(ToElements(name));
            return _nullConstant;
        }

        /// <summary>
        /// <para>
        /// Creates the element using the specified name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="name">
        /// <para>The name.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

```
        /// <para>The null constant.</para>
        /// <para></para>
        /// </returns>
        public TLink CreateElement(string name)
        {
            _index.Add(ToElements(name));
            return _nullConstant;
        }

        /// <summary>
        /// <para>
        /// Creates the text element using the specified content.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="content">
        /// <para>The content.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The null constant.</para>
        /// <para></para>
        /// </returns>
        public TLink CreateTextElement(string content)
        {
            _index.Add(ToElements(content));
            return _nullConstant;
        }

        /// <summary>
        /// <para>
        /// Gets the document using the specified name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="name">
        /// <para>The name.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="System.NotImplementedException">
        /// <para></para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetDocument(string name)
        {
            throw new System.NotImplementedException();
        }

        /// <summary>
        /// <para>
        /// Gets the element using the specified name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="name">
        /// <para>The name.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="System.NotImplementedException">
        /// <para></para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetElement(string name)
        {
            throw new System.NotImplementedException();
        }

        /// <summary>
        /// <para>
        /// Gets the text element using the specified content.
```

```
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="content">
        /// <para>The content.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="System.NotImplementedException">
        /// <para></para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetTextElement(string content)
        {
            throw new System.NotImplementedException();
        }

        /// <summary>
        /// <para>
        /// Gets the children using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="System.NotImplementedException">
        /// <para></para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>A list of i list t link</para>
        /// <para></para>
        /// </returns>
        public IList<TLink> GetChildren(TLink parent)
        {
            throw new System.NotImplementedException();
        }
    }
}
```

Index