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

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

```csharp
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(markerIndex));
            _unicodeSequenceMarker = links.GetOrCreate(meaningRoot,
             ↪  Arithmetic.Increment(markerIndex));
            _elementMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(markerIndex));
            _textElementMarker = links.GetOrCreate(meaningRoot,
             ↪  Arithmetic.Increment(markerIndex));
            _documentMarker = links.GetOrCreate(meaningRoot, Arithmetic.Increment(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);

        private TLink Create(TLink marker, string content)
        {
            var contentSequence = _stringToUnicodeSequenceConverter.Convert(content);
            return _links.GetOrCreate(marker, contentSequence);
        }

        public void AttachElementToParent(TLink elementToAttach, TLink parent) =>
         ↪  _links.GetOrCreate(parent, elementToAttach);
    }
}
```

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
{
    public interface ICommandLineInterface
    {
        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

namespace Platform.Data.Doublets.Xml
{
    public interface IXmlStorage<TLink>
    {
        TLink CreateDocument(string name);
        TLink CreateElement(string name);
        TLink CreateTextElement(string content);
        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
{
    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
{
    public class XmlElementCounter
    {
        public XmlElementCounter() { }

        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
            {
                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)
        {
            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;
            }
        }
    }
}
```

1.8 ./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
{
    public class XmlImporterCLI : ICommandLineInterface
    {
        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();
                    }
                }
            }
        }
    }
}
```

1.9 ./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
{
    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

./csharp/Platform.Data.Doublets.Xml/DefaultXmlStorage.cs, 1
./csharp/Platform.Data.Doublets.Xml/ICommandLineInterface.cs, 2
./csharp/Platform.Data.Doublets.Xml/IXmlStorage.cs, 2
./csharp/Platform.Data.Doublets.Xml/XmlElementContext.cs, 2
./csharp/Platform.Data.Doublets.Xml/XmlElementCounter.cs, 2
./csharp/Platform.Data.Doublets.Xml/XmlElementCounterCLI.cs, 4
./csharp/Platform.Data.Doublets.Xml/XmlImporter.cs, 4
./csharp/Platform.Data.Doublets.Xml/XmlImporterCLI.cs, 6
./csharp/Platform.Data.Doublets.Xml/XmlIndexer.cs, 6