1.1 ./csharp/Platform.Data.Doublets.Json/DefaultJsonStorage.cs

```
using Platform.Numbers;
using Platform.Data.Doublets.Unicode;
using Platform.Data.Doublets.Sequences.Converters;
using Platform.Data.Doublets.CriterionMatchers;
using Platform.Data.Numbers.Raw;
using Platform.Converters;
using Platform.Data.Doublets.Sequences.Walkers;
using Platform.Collections.Stacks;
using System;
using System.Collections.Generic;
using Platform.Data.Doublets.Numbers.Rational;
using Platform.Data.Doublets.Numbers.Raw;
using Platform.Data.Doublets.Sequences.HeightProviders;
using Platform.Data.Doublets.Sequences;

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

namespace Platform.Data.Doublets.Json
{
    /// <summary>
    /// <para>
    /// Represents the default json storage.
    /// </para>
    /// <para></para>
    /// </summary>
    /// <seealso cref="IJsonStorage{TLink}"/>
    public class DefaultJsonStorage<TLink> : IJsonStorage<TLink>
        where TLink : struct
    {
        /// <summary>
        /// <para>
        /// The any.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly TLink Any;
        /// <summary>
        /// <para>
        /// The zero.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly TLink Zero = default;
        /// <summary>
        /// <para>
        /// The zero.
        /// </para>
        /// <para></para>
        /// </summary>
        public static readonly TLink One = Arithmetic.Increment(Zero);
        /// <summary>
        /// <para>
        /// The balanced variant converter.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly BalancedVariantConverter<TLink> BalancedVariantConverter;
        /// <summary>
        /// <para>
        /// The list to sequence converter.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly IConverter<IList<TLink>, TLink> ListToSequenceConverter;
        /// <summary>
        /// <para>
        /// The meaning root.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly TLink MeaningRoot;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
```

```
78          public readonly EqualityComparer<TLink> EqualityComparer =
            ↪ EqualityComparer<TLink>.Default;
79          // Converters that are able to convert link's address (UInt64 value) to a raw number
            ↪ represented with another UInt64 value and back
80          /// <summary>
81          /// <para>
82          /// The number to address converter.
83          /// </para>
84          /// <para></para>
85          /// </summary>
86          public readonly RawNumberToAddressConverter<TLink> NumberToAddressConverter = new();
87          /// <summary>
88          /// <para>
89          /// The address to number converter.
90          /// </para>
91          /// <para></para>
92          /// </summary>
93          public readonly AddressToRawNumberConverter<TLink> AddressToNumberConverter = new();
94          // Converters between BigInteger and raw number sequence
95          /// <summary>
96          /// <para>
97          /// The big integer to raw number sequence converter.
98          /// </para>
99          /// <para></para>
100         /// </summary>
101         public readonly BigIntegerToRawNumberSequenceConverter<TLink>
            ↪ BigIntegerToRawNumberSequenceConverter;
102         /// <summary>
103         /// <para>
104         /// The raw number sequence to big integer converter.
105         /// </para>
106         /// <para></para>
107         /// </summary>
108         public readonly RawNumberSequenceToBigIntegerConverter<TLink>
            ↪ RawNumberSequenceToBigIntegerConverter;
109         // Converters between decimal and rational number sequence
110         /// <summary>
111         /// <para>
112         /// The decimal to rational converter.
113         /// </para>
114         /// <para></para>
115         /// </summary>
116         public readonly DecimalToRationalConverter<TLink> DecimalToRationalConverter;
117         /// <summary>
118         /// <para>
119         /// The rational to decimal converter.
120         /// </para>
121         /// <para></para>
122         /// </summary>
123         public readonly RationalToDecimalConverter<TLink> RationalToDecimalConverter;
124         // Converters between string and unicode sequence
125         /// <summary>
126         /// <para>
127         /// The string to unicode sequence converter.
128         /// </para>
129         /// <para></para>
130         /// </summary>
131         public readonly IConverter<string, TLink> StringToUnicodeSequenceConverter;
132         /// <summary>
133         /// <para>
134         /// The unicode sequence to string converter.
135         /// </para>
136         /// <para></para>
137         /// </summary>
138         public readonly IConverter<TLink, string> UnicodeSequenceToStringConverter;
139         // For sequences
140         /// <summary>
141         /// <para>
142         /// The json array element criterion matcher.
143         /// </para>
144         /// <para></para>
145         /// </summary>
146         public readonly JsonArrayElementCriterionMatcher<TLink> JsonArrayElementCriterionMatcher;
147         /// <summary>
148         /// <para>
149         /// The default sequence right height provider.
150         /// </para>
151         /// <para></para>
```

```
152         /// </summary>
153         public readonly DefaultSequenceRightHeightProvider<TLink>
            ↪  DefaultSequenceRightHeightProvider;
154         /// <summary>
155         /// <para>
156         /// The default sequence appender.
157         /// </para>
158         /// <para></para>
159         /// </summary>
160         public readonly DefaultSequenceAppender<TLink> DefaultSequenceAppender;
161         /// <summary>
162         /// <para>
163         /// Gets the links value.
164         /// </para>
165         /// <para></para>
166         /// </summary>
167         public ILinks<TLink> Links { get; }
168         /// <summary>
169         /// <para>
170         /// Gets the document marker value.
171         /// </para>
172         /// <para></para>
173         /// </summary>
174         public TLink DocumentMarker { get; }
175         /// <summary>
176         /// <para>
177         /// Gets the object marker value.
178         /// </para>
179         /// <para></para>
180         /// </summary>
181         public TLink ObjectMarker { get; }
182         /// <summary>
183         /// <para>
184         /// Gets the member marker value.
185         /// </para>
186         /// <para></para>
187         /// </summary>
188         public TLink MemberMarker { get; }
189         /// <summary>
190         /// <para>
191         /// Gets the value marker value.
192         /// </para>
193         /// <para></para>
194         /// </summary>
195         public TLink ValueMarker { get; }
196         /// <summary>
197         /// <para>
198         /// Gets the string marker value.
199         /// </para>
200         /// <para></para>
201         /// </summary>
202         public TLink StringMarker { get; }
203         /// <summary>
204         /// <para>
205         /// Gets the empty string marker value.
206         /// </para>
207         /// <para></para>
208         /// </summary>
209         public TLink EmptyStringMarker { get; }
210         /// <summary>
211         /// <para>
212         /// Gets the number marker value.
213         /// </para>
214         /// <para></para>
215         /// </summary>
216         public TLink NumberMarker { get; }
217         /// <summary>
218         /// <para>
219         /// Gets the negative number marker value.
220         /// </para>
221         /// <para></para>
222         /// </summary>
223         public TLink NegativeNumberMarker { get; }
224         /// <summary>
225         /// <para>
226         /// Gets the array marker value.
227         /// </para>
228         /// <para></para>
```

```
229         /// </summary>
230         public TLink ArrayMarker { get; }
231         /// <summary>
232         /// <para>
233         /// Gets the empty array marker value.
234         /// </para>
235         /// <para></para>
236         /// </summary>
237         public TLink EmptyArrayMarker { get; }
238         /// <summary>
239         /// <para>
240         /// Gets the true marker value.
241         /// </para>
242         /// <para></para>
243         /// </summary>
244         public TLink TrueMarker { get; }
245         /// <summary>
246         /// <para>
247         /// Gets the false marker value.
248         /// </para>
249         /// <para></para>
250         /// </summary>
251         public TLink FalseMarker { get; }
252         /// <summary>
253         /// <para>
254         /// Gets the null marker value.
255         /// </para>
256         /// <para></para>
257         /// </summary>
258         public TLink NullMarker { get; }
259
260         /// <summary>
261         /// <para>
262         /// Initializes a new <see cref="DefaultJsonStorage"/> instance.
263         /// </para>
264         /// <para></para>
265         /// </summary>
266         /// <param name="links">
267         /// <para>A links.</para>
268         /// <para></para>
269         /// </param>
270         /// <param name="listToSequenceConverter">
271         /// <para>A list to sequence converter.</para>
272         /// <para></para>
273         /// </param>
274         public DefaultJsonStorage(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
            ↪ listToSequenceConverter)
275         {
276             Links = links;
277             ListToSequenceConverter = listToSequenceConverter;
278             // Initializes constants
279             Any = Links.Constants.Any;
280             var markerIndex = One;
281             MeaningRoot = links.GetOrCreate(markerIndex, markerIndex);
282             var unicodeSymbolMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
283             var unicodeSequenceMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
284             DocumentMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
285             ObjectMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
286             MemberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
287             ValueMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
288             StringMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
289             EmptyStringMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
290             NumberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
291             NegativeNumberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
292             ArrayMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
293             EmptyArrayMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
                ↪ markerIndex));
294             TrueMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
295             FalseMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
296             NullMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
297             BalancedVariantConverter = new(links);
298             TargetMatcher<TLink> unicodeSymbolCriterionMatcher = new(Links, unicodeSymbolMarker);
```

```
            TargetMatcher<TLink> unicodeSequenceCriterionMatcher = new(Links,
            ↪  unicodeSequenceMarker);
            CharToUnicodeSymbolConverter<TLink> charToUnicodeSymbolConverter =
                new(Links, AddressToNumberConverter, unicodeSymbolMarker);
            UnicodeSymbolToCharConverter<TLink> unicodeSymbolToCharConverter =
                new(Links, NumberToAddressConverter, unicodeSymbolCriterionMatcher);
            StringToUnicodeSequenceConverter = new CachingConverterDecorator<string, TLink>(
                new StringToUnicodeSequenceConverter<TLink>(Links, charToUnicodeSymbolConverter,
                    BalancedVariantConverter, unicodeSequenceMarker));
            RightSequenceWalker<TLink> sequenceWalker =
                new(Links, new DefaultStack<TLink>(), unicodeSymbolCriterionMatcher.IsMatched);
            UnicodeSequenceToStringConverter = new CachingConverterDecorator<TLink, string>(
                new UnicodeSequenceToStringConverter<TLink>(Links,
                ↪  unicodeSequenceCriterionMatcher, sequenceWalker,
                    unicodeSymbolToCharConverter));
            BigIntegerToRawNumberSequenceConverter =
                new(links, AddressToNumberConverter, ListToSequenceConverter,
                ↪  NegativeNumberMarker);
            RawNumberSequenceToBigIntegerConverter = new(links, NumberToAddressConverter,
            ↪  NegativeNumberMarker);
            DecimalToRationalConverter = new(links, BigIntegerToRawNumberSequenceConverter);
            RationalToDecimalConverter = new(links, RawNumberSequenceToBigIntegerConverter);
            JsonArrayElementCriterionMatcher = new(this);
            DefaultSequenceRightHeightProvider = new(Links, JsonArrayElementCriterionMatcher);
            DefaultSequenceAppender = new(Links, new DefaultStack<TLink>(),
            ↪  DefaultSequenceRightHeightProvider);
        }

        /// <summary>
        /// <para>
        /// Creates the string 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 CreateString(string content)
        {
            var @string = GetStringSequence(content);
            return Links.GetOrCreate(StringMarker, @string);
        }

        /// <summary>
        /// <para>
        /// Creates the string value 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 CreateStringValue(string content)
        {
            var @string = CreateString(content);
            return CreateValue(@string);
        }

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

```
372         /// <returns>
373         /// <para>The link</para>
374         /// <para></para>
375         /// </returns>
376         public TLink CreateNumber(decimal number)
377         {
378             var numberSequence = DecimalToRationalConverter.Convert(number);
379             return Links.GetOrCreate(NumberMarker, numberSequence);
380         }
381 
382         /// <summary>
383         /// <para>
384         /// Creates the number value using the specified number.
385         /// </para>
386         /// <para></para>
387         /// </summary>
388         /// <param name="number">
389         /// <para>The number.</para>
390         /// <para></para>
391         /// </param>
392         /// <returns>
393         /// <para>The link</para>
394         /// <para></para>
395         /// </returns>
396         public TLink CreateNumberValue(decimal number)
397         {
398             var numberLink = CreateNumber(number);
399             return CreateValue(numberLink);
400         }
401 
402         /// <summary>
403         /// <para>
404         /// Creates the boolean value using the specified value.
405         /// </para>
406         /// <para></para>
407         /// </summary>
408         /// <param name="value">
409         /// <para>The value.</para>
410         /// <para></para>
411         /// </param>
412         /// <returns>
413         /// <para>The link</para>
414         /// <para></para>
415         /// </returns>
416         public TLink CreateBooleanValue(bool value) => CreateValue(value ? TrueMarker :
            ↪ FalseMarker);
417 
418         /// <summary>
419         /// <para>
420         /// Creates the null value.
421         /// </para>
422         /// <para></para>
423         /// </summary>
424         /// <returns>
425         /// <para>The link</para>
426         /// <para></para>
427         /// </returns>
428         public TLink CreateNullValue() => CreateValue(NullMarker);
429 
430         /// <summary>
431         /// <para>
432         /// Creates the document using the specified name.
433         /// </para>
434         /// <para></para>
435         /// </summary>
436         /// <param name="name">
437         /// <para>The name.</para>
438         /// <para></para>
439         /// </param>
440         /// <returns>
441         /// <para>The link</para>
442         /// <para></para>
443         /// </returns>
444         public TLink CreateDocument(string name)
445         {
446             var documentName = CreateString(name);
447             return Links.GetOrCreate(DocumentMarker, documentName);
448         }
```

```
449
450         /// <summary>
451         /// <para>
452         /// Creates the object.
453         /// </para>
454         /// <para></para>
455         /// </summary>
456         /// <returns>
457         /// <para>The link</para>
458         /// <para></para>
459         /// </returns>
460         public TLink CreateObject()
461         {
462             var @object = Links.Create();
463             return Links.Update(@object, newSource: ObjectMarker, newTarget: @object);
464         }
465
466         /// <summary>
467         /// <para>
468         /// Creates the object value.
469         /// </para>
470         /// <para></para>
471         /// </summary>
472         /// <returns>
473         /// <para>The link</para>
474         /// <para></para>
475         /// </returns>
476         public TLink CreateObjectValue()
477         {
478             var @object = CreateObject();
479             return CreateValue(@object);
480         }
481
482         /// <summary>
483         /// <para>
484         /// Creates the array using the specified array.
485         /// </para>
486         /// <para></para>
487         /// </summary>
488         /// <param name="array">
489         /// <para>The array.</para>
490         /// <para></para>
491         /// </param>
492         /// <returns>
493         /// <para>The link</para>
494         /// <para></para>
495         /// </returns>
496         public TLink CreateArray(IList<TLink> array)
497         {
498             var arraySequence = array.Count == 0 ? EmptyArrayMarker :
                ↪  BalancedVariantConverter.Convert(array);
499             return CreateArray(arraySequence);
500         }
501
502         /// <summary>
503         /// <para>
504         /// Creates the array using the specified sequence.
505         /// </para>
506         /// <para></para>
507         /// </summary>
508         /// <param name="sequence">
509         /// <para>The sequence.</para>
510         /// <para></para>
511         /// </param>
512         /// <returns>
513         /// <para>The link</para>
514         /// <para></para>
515         /// </returns>
516         public TLink CreateArray(TLink sequence) => Links.GetOrCreate(ArrayMarker, sequence);
517
518         /// <summary>
519         /// <para>
520         /// Creates the array value using the specified array.
521         /// </para>
522         /// <para></para>
523         /// </summary>
524         /// <param name="array">
525         /// <para>The array.</para>
```

```
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink CreateArrayValue(IList<TLink> array)
        {
            var arrayLink = CreateArray(array);
            return CreateValue(arrayLink);
        }

        /// <summary>
        /// <para>
        /// Creates the array value using the specified sequence.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="sequence">
        /// <para>The sequence.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink CreateArrayValue(TLink sequence)
        {
            var array = CreateArray(sequence);
            return CreateValue(array);
        }

        /// <summary>
        /// <para>
        /// Creates the member 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 CreateMember(string name)
        {
            var nameLink = CreateString(name);
            return Links.GetOrCreate(MemberMarker, nameLink);
        }

        /// <summary>
        /// <para>
        /// Creates the value using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink CreateValue(TLink value) => Links.GetOrCreate(ValueMarker, value);

        /// <summary>
        /// <para>
        /// Attaches the object using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
```

```
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink AttachObject(TLink parent) => Attach(parent, CreateObjectValue());

        /// <summary>
        /// <para>
        /// Attaches the string using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <param name="content">
        /// <para>The content.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink AttachString(TLink parent, string content)
        {
            var @string = CreateString(content);
            var stringValue = CreateValue(@string);
            return Attach(parent, stringValue);
        }

        /// <summary>
        /// <para>
        /// Attaches the number using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <param name="number">
        /// <para>The number.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink AttachNumber(TLink parent, decimal number)
        {
            var numberLink = CreateNumber(number);
            var numberValue = CreateValue(numberLink);
            return Attach(parent, numberValue);
        }

        /// <summary>
        /// <para>
        /// Attaches the boolean using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink AttachBoolean(TLink parent, bool value)
        {
            var booleanValue = CreateBooleanValue(value);
            return Attach(parent, booleanValue);
```

```
682         }
683
684         /// <summary>
685         /// <para>
686         /// Attaches the null using the specified parent.
687         /// </para>
688         /// <para></para>
689         /// </summary>
690         /// <param name="parent">
691         /// <para>The parent.</para>
692         /// <para></para>
693         /// </param>
694         /// <returns>
695         /// <para>The link</para>
696         /// <para></para>
697         /// </returns>
698         public TLink AttachNull(TLink parent)
699         {
700             var nullValue = CreateNullValue();
701             return Attach(parent, nullValue);
702         }
703
704         /// <summary>
705         /// <para>
706         /// Attaches the array using the specified parent.
707         /// </para>
708         /// <para></para>
709         /// </summary>
710         /// <param name="parent">
711         /// <para>The parent.</para>
712         /// <para></para>
713         /// </param>
714         /// <param name="array">
715         /// <para>The array.</para>
716         /// <para></para>
717         /// </param>
718         /// <returns>
719         /// <para>The link</para>
720         /// <para></para>
721         /// </returns>
722         public TLink AttachArray(TLink parent, IList<TLink> array)
723         {
724             var arrayValue = CreateArrayValue(array);
725             return Attach(parent, arrayValue);
726         }
727
728         /// <summary>
729         /// <para>
730         /// Attaches the member to object using the specified object.
731         /// </para>
732         /// <para></para>
733         /// </summary>
734         /// <param name="@object">
735         /// <para>The object.</para>
736         /// <para></para>
737         /// </param>
738         /// <param name="keyName">
739         /// <para>The key name.</para>
740         /// <para></para>
741         /// </param>
742         /// <returns>
743         /// <para>The link</para>
744         /// <para></para>
745         /// </returns>
746         public TLink AttachMemberToObject(TLink @object, string keyName)
747         {
748             var member = CreateMember(keyName);
749             return Attach(@object, member);
750         }
751
752         /// <summary>
753         /// <para>
754         /// Attaches the parent.
755         /// </para>
756         /// <para></para>
757         /// </summary>
758         /// <param name="parent">
759         /// <para>The parent.</para>
```

```
        /// <para></para>
        /// </param>
        /// <param name="child">
        /// <para>The child.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink Attach(TLink parent, TLink child) => Links.GetOrCreate(parent, child);

        /// <summary>
        /// <para>
        /// Appends the array value using the specified array value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="arrayValue">
        /// <para>The array value.</para>
        /// <para></para>
        /// </param>
        /// <param name="appendant">
        /// <para>The appendant.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The new array value.</para>
        /// <para></para>
        /// </returns>
        public TLink AppendArrayValue(TLink arrayValue, TLink appendant)
        {
            var array = GetArray(arrayValue);
            var arraySequence = Links.GetTarget(array);
            TLink newArrayValue;
            if (EqualityComparer.Equals(arraySequence, EmptyArrayMarker))
            {
                newArrayValue = CreateArrayValue(appendant);
            }
            else
            {
                arraySequence = DefaultSequenceAppender.Append(arraySequence, appendant);
                newArrayValue = CreateArrayValue(arraySequence);
            }
            return newArrayValue;
        }

        /// <summary>
        /// <para>
        /// Gets the document or default 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 GetDocumentOrDefault(string name)
        {
            var stringSequence = GetStringSequence(name);
            var @string = Links.SearchOrDefault(StringMarker, stringSequence);
            if (EqualityComparer.Equals(@string, default))
            {
                return default;
            }
            return Links.SearchOrDefault(DocumentMarker, @string);
        }

        /// <summary>
        /// <para>
        /// Gets the string sequence 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>
        private TLink GetStringSequence(string content) => content == "" ? EmptyStringMarker :
        ↪ StringToUnicodeSequenceConverter.Convert(content);

        /// <summary>
        /// <para>
        /// Gets the string using the specified string value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="stringValue">
        /// <para>The string value.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="Exception">
        /// <para>The passed link does not contain a string.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        public string GetString(TLink stringValue)
        {
            var current = stringValue;
            TLink source;
            for (int i = 0; i < 3; i++)
            {
                source = Links.GetSource(current);
                if (EqualityComparer.Equals(source, StringMarker))
                {
                    var sequence = Links.GetTarget(current);
                    var isEmpty = EqualityComparer.Equals(sequence, EmptyStringMarker);
                    return isEmpty ? "" : UnicodeSequenceToStringConverter.Convert(sequence);
                }
                current = Links.GetTarget(current);
            }
            throw new Exception("The passed link does not contain a string.");
        }

        /// <summary>
        /// <para>
        /// Gets the number using the specified value link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="valueLink">
        /// <para>The value link.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="Exception">
        /// <para>The passed link does not contain a number.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The decimal</para>
        /// <para></para>
        /// </returns>
        public decimal GetNumber(TLink valueLink)
        {
            var current = valueLink;
            TLink source;
            TLink target;
            for (int i = 0; i < 3; i++)
            {
                source = Links.GetSource(current);
                target = Links.GetTarget(current);
                if (EqualityComparer.Equals(source, NumberMarker))
                {
                    return RationalToDecimalConverter.Convert(target);
                }
```

```
                current = target;
            }
            throw new Exception("The passed link does not contain a number.");
        }


        /// <summary>
        /// <para>
        /// Gets the object using the specified object value link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="objectValueLink">
        /// <para>The object value link.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="Exception">
        /// <para>The passed link does not contain an object.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetObject(TLink objectValueLink)
        {
            var current = objectValueLink;
            TLink source;
            for (int i = 0; i < 3; i++)
            {
                source = Links.GetSource(current);
                if (EqualityComparer.Equals(source, ObjectMarker))
                {
                    return current;
                }
                current = Links.GetTarget(current);
            }
            throw new Exception("The passed link does not contain an object.");
        }

        /// <summary>
        /// <para>
        /// Gets the array using the specified array value link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="arrayValueLink">
        /// <para>The array value link.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="Exception">
        /// <para>The passed link does not contain an array.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetArray(TLink arrayValueLink)
        {
            var current = arrayValueLink;
            TLink source;
            for (int i = 0; i < 3; i++)
            {
                source = Links.GetSource(current);
                if (EqualityComparer.Equals(source, ArrayMarker))
                {
                    return current;
                }
                current = Links.GetTarget(current);
            }
            throw new Exception("The passed link does not contain an array.");
        }

        /// <summary>
        /// <para>
        /// Gets the array sequence using the specified array.
        /// </para>
```

```
        /// <para></para>
        /// </summary>
        /// <param name="array">
        /// <para>The array.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetArraySequence(TLink array) => Links.GetTarget(array);

        /// <summary>
        /// <para>
        /// Gets the value link using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="InvalidOperationException">
        /// <para>More than 1 value found.</para>
        /// <para></para>
        /// </exception>
        /// <exception cref="InvalidOperationException">
        /// <para>The list elements length is negative.</para>
        /// <para></para>
        /// </exception>
        /// <exception cref="InvalidOperationException">
        /// <para>The passed link is not a value.</para>
        /// <para></para>
        /// </exception>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        public TLink GetValueLink(TLink parent)
        {
            var query = new Link<TLink>(index: Any, source: parent, target: Any);
            var resultLinks = Links.All(query);
            switch (resultLinks.Count)
            {
                case 0:
                    return default;
                case 1:
                    var resultLinkTarget = Links.GetTarget(resultLinks[0]);
                    if (EqualityComparer.Equals(Links.GetSource(resultLinkTarget), ValueMarker))
                    {
                        return resultLinkTarget;
                    }
                    else
                    {
                        throw new InvalidOperationException("The passed link is not a value.");
                    }
                case > 1:
                    throw new InvalidOperationException("More than 1 value found.");
                default:
                    throw new InvalidOperationException("The list elements length is negative.");
            }
        }

        /// <summary>
        /// <para>
        /// Gets the value marker using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The target source.</para>
        /// <para></para>
        /// </returns>
        public TLink GetValueMarker(TLink value)
```

```
        {
            var target = Links.GetTarget(value);
            var targetSource = Links.GetSource(target);
            if (EqualityComparer.Equals(MeaningRoot, targetSource))
            {
                return target;
            }
            return targetSource;
        }

        /// <summary>
        /// <para>
        /// Gets the members links using the specified object.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@object">
        /// <para>The object.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The members.</para>
        /// <para></para>
        /// </returns>
        public List<TLink> GetMembersLinks(TLink @object)
        {
            Link<TLink> query = new(index: Any, source: @object, target: Any);
            List<TLink> members = new();
            Links.Each(objectMemberLink =>
            {
                var memberLink = Links.GetTarget(objectMemberLink);
                var memberMarker = Links.GetSource(memberLink);
                if (EqualityComparer.Equals(memberMarker, MemberMarker))
                {
                    members.Add(Links.GetIndex(objectMemberLink));
                }
                return Links.Constants.Continue;
            }, query);
            return members;
        }
    }
}
```

1.2 ./csharp/Platform.Data.Doublets.Json/IJsonStorage.cs

```
using System.Collections.Generic;

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

namespace Platform.Data.Doublets.Json
{
    /// <summary>
    /// <para>
    /// Defines the json storage.
    /// </para>
    /// <para></para>
    /// </summary>
    public interface IJsonStorage<TLink>
    {
        /// <summary>
        /// <para>
        /// Gets the links value.
        /// </para>
        /// <para></para>
        /// </summary>
        public ILinks<TLink> Links { get; }
        /// <summary>
        /// <para>
        /// Gets the document marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink DocumentMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the object marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink ObjectMarker { get; }
```

```
        /// <summary>
        /// <para>
        /// Gets the string marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink StringMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the empty string marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink EmptyStringMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the member marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink MemberMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the value marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink ValueMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the number marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink NumberMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the array marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink ArrayMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the empty array marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink EmptyArrayMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the true marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink TrueMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the false marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink FalseMarker { get; }
        /// <summary>
        /// <para>
        /// Gets the null marker value.
        /// </para>
        /// <para></para>
        /// </summary>
        public TLink NullMarker { get; }
        /// <summary>
        /// <para>
        /// Creates the string 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 CreateString(string content);
        /// <summary>
        /// <para>
        /// Creates the string value 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 CreateStringValue(string content);
        /// <summary>
        /// <para>
        /// Creates the number using the specified number.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="number">
        /// <para>The number.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateNumber(decimal number);
        /// <summary>
        /// <para>
        /// Creates the number value using the specified number.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="number">
        /// <para>The number.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateNumberValue(decimal number);
        /// <summary>
        /// <para>
        /// Creates the boolean value using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateBooleanValue(bool value);
        /// <summary>
        /// <para>
        /// Creates the null value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateNullValue();
```

```
        /// <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 link</para>
        /// <para></para>
        /// </returns>
        TLink CreateDocument(string name);
        /// <summary>
        /// <para>
        /// Gets the document or default 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>
        TLink GetDocumentOrDefault(string name);
        /// <summary>
        /// <para>
        /// Creates the object.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateObject();
        /// <summary>
        /// <para>
        /// Creates the object value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateObjectValue();
        /// <summary>
        /// <para>
        /// Creates the array using the specified array.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="array">
        /// <para>The array.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateArray(IList<TLink> array);
        /// <summary>
        /// <para>
        /// Creates the array value using the specified array.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="array">
        /// <para>The array.</para>
        /// <para></para>
        /// </param>
        /// <returns>
```

```
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateArrayValue(IList<TLink> array) => CreateValue(CreateArray(array));
        /// <summary>
        /// <para>
        /// Creates the array value using the specified array.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="array">
        /// <para>The array.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateArrayValue(TLink array) => CreateValue(array);
        /// <summary>
        /// <para>
        /// Creates the member 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>
        TLink CreateMember(string name);
        /// <summary>
        /// <para>
        /// Creates the value using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink CreateValue(TLink value);
        /// <summary>
        /// <para>
        /// Attaches the source.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="source">
        /// <para>The source.</para>
        /// <para></para>
        /// </param>
        /// <param name="target">
        /// <para>The target.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink Attach(TLink source, TLink target);
        /// <summary>
        /// <para>
        /// Attaches the object using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
```

```
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink AttachObject(TLink parent);
        /// <summary>
        /// <para>
        /// Attaches the string using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <param name="content">
        /// <para>The content.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink AttachString(TLink parent, string content);
        /// <summary>
        /// <para>
        /// Attaches the number using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <param name="number">
        /// <para>The number.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink AttachNumber(TLink parent, decimal number);
        /// <summary>
        /// <para>
        /// Attaches the boolean using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <param name="value">
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink AttachBoolean(TLink parent, bool value);
        /// <summary>
        /// <para>
        /// Attaches the null using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink AttachNull(TLink parent);
        /// <summary>
```

```
        /// <para>
        /// Attaches the array using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <param name="array">
        /// <para>The array.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink AttachArray(TLink parent, IList<TLink> array);
        /// <summary>
        /// <para>
        /// Attaches the member to object using the specified object.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="@object">
        /// <para>The object.</para>
        /// <para></para>
        /// </param>
        /// <param name="keyName">
        /// <para>The key name.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink AttachMemberToObject(TLink @object, string keyName);
        /// <summary>
        /// <para>
        /// Appends the array value using the specified array value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="arrayValue">
        /// <para>The array value.</para>
        /// <para></para>
        /// </param>
        /// <param name="appendant">
        /// <para>The appendant.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink AppendArrayValue(TLink arrayValue, TLink appendant);
        /// <summary>
        /// <para>
        /// Gets the string using the specified string value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="stringValue">
        /// <para>The string value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The string</para>
        /// <para></para>
        /// </returns>
        string GetString(TLink stringValue);
        /// <summary>
        /// <para>
        /// Gets the number using the specified value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="value">
```

```
        /// <para>The value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The decimal</para>
        /// <para></para>
        /// </returns>
        decimal GetNumber(TLink value);
        /// <summary>
        /// <para>
        /// Gets the object using the specified object value.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="objectValue">
        /// <para>The object value.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink GetObject(TLink objectValue);
        /// <summary>
        /// <para>
        /// Gets the array using the specified array value link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="arrayValueLink">
        /// <para>The array value link.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink GetArray(TLink arrayValueLink);
        /// <summary>
        /// <para>
        /// Gets the array sequence using the specified array.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="array">
        /// <para>The array.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink GetArraySequence(TLink array);
        /// <summary>
        /// <para>
        /// Gets the value link using the specified parent.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="parent">
        /// <para>The parent.</para>
        /// <para></para>
        /// </param>
        /// <returns>
        /// <para>The link</para>
        /// <para></para>
        /// </returns>
        TLink GetValueLink(TLink parent);
        /// <summary>
        /// <para>
        /// Gets the value marker using the specified link.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="link">
        /// <para>The link.</para>
        /// <para></para>
        /// </param>
```

```
582             /// <returns>
583             /// <para>The link</para>
584             /// <para></para>
585             /// </returns>
586             TLink GetValueMarker(TLink link);
587             /// <summary>
588             /// <para>
589             /// Gets the members links using the specified object.
590             /// </para>
591             /// <para></para>
592             /// </summary>
593             /// <param name="@object">
594             /// <para>The object.</para>
595             /// <para></para>
596             /// </param>
597             /// <returns>
598             /// <para>A list of t link</para>
599             /// <para></para>
600             /// </returns>
601             List<TLink> GetMembersLinks(TLink @object);
602         }
603     }
```

1.3 ./csharp/Platform.Data.Doublets.Json/JsonArrayElementCriterionMatcher.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6   using System.Text.Json;
7   using System.Threading;
8   using System.IO;
9   using Platform.Converters;
10  using System.Collections;
11  using Platform.Data.Doublets.Sequences;
12  using Platform.Data.Doublets.Sequences.HeightProviders;
13  using Platform.Data.Doublets.Sequences.CriterionMatchers;
14  using Platform.Interfaces;
15
16  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18  namespace Platform.Data.Doublets.Json
19  {
20      /// <summary>
21      /// <para>
22      /// Represents the json array element criterion matcher.
23      /// </para>
24      /// <para></para>
25      /// </summary>
26      /// <seealso cref="ICriterionMatcher{TLink}"/>
27      public class JsonArrayElementCriterionMatcher<TLink> : ICriterionMatcher<TLink>
28      {
29          /// <summary>
30          /// <para>
31          /// The storage.
32          /// </para>
33          /// <para></para>
34          /// </summary>
35          public readonly IJsonStorage<TLink> Storage;
36          /// <summary>
37          /// <para>
38          /// Initializes a new <see cref="JsonArrayElementCriterionMatcher"/> instance.
39          /// </para>
40          /// <para></para>
41          /// </summary>
42          /// <param name="storage">
43          /// <para>A storage.</para>
44          /// <para></para>
45          /// </param>
46          public JsonArrayElementCriterionMatcher(IJsonStorage<TLink> storage) => Storage =
            ↪  storage;
47          /// <summary>
48          /// <para>
49          /// Determines whether this instance is matched.
50          /// </para>
51          /// <para></para>
52          /// </summary>
53          /// <param name="link">
54          /// <para>The link.</para>
```

```
55          /// <para></para>
56          /// </param>
57          /// <returns>
58          /// <para>The bool</para>
59          /// <para></para>
60          /// </returns>
61          public bool IsMatched(TLink link) =>
            ↪  EqualityComparer<TLink>.Default.Equals(Storage.Links.GetSource(link),
            ↪  Storage.ValueMarker);
62      }
63  }
```

1.4 ./csharp/Platform.Data.Doublets.Json/JsonExporter.cs

```
1   using System;
2   using System.Collections.Generic;
3   using System.Text.Json;
4   using System.Threading;
5   using Platform.Data.Doublets.Sequences.Walkers;
6   using Platform.Collections.Stacks;
7
8   #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
9
10  namespace Platform.Data.Doublets.Json
11  {
12      /// <summary>
13      /// <para>
14      /// Represents the json exporter.
15      /// </para>
16      /// <para></para>
17      /// </summary>
18      public class JsonExporter<TLink>
19      {
20          /// <summary>
21          /// <para>
22          /// The storage.
23          /// </para>
24          /// <para></para>
25          /// </summary>
26          public readonly IJsonStorage<TLink> Storage;
27          /// <summary>
28          /// <para>
29          /// The default.
30          /// </para>
31          /// <para></para>
32          /// </summary>
33          public readonly EqualityComparer<TLink> EqualityComparer =
            ↪  EqualityComparer<TLink>.Default;
34
35          /// <summary>
36          /// <para>
37          /// Initializes a new <see cref="JsonExporter"/> instance.
38          /// </para>
39          /// <para></para>
40          /// </summary>
41          /// <param name="storage">
42          /// <para>A storage.</para>
43          /// <para></para>
44          /// </param>
45          public JsonExporter(IJsonStorage<TLink> storage) => Storage = storage;
46
47              /// <summary>
48              /// <para>
49              /// Determines whether this instance is element.
50              /// </para>
51              /// <para></para>
52              /// </summary>
53              /// <param name="link">
54              /// <para>The link.</para>
55              /// <para></para>
56              /// </param>
57              /// <returns>
58              /// <para>The bool</para>
59              /// <para></para>
60              /// </returns>
61              private bool IsElement(TLink link)
62          {
63              var marker = Storage.Links.GetSource(link);
64              return EqualityComparer.Equals(marker, Storage.ValueMarker);
65          }
```

```
66
67          /// <summary>
68          /// <para>
69          /// Writes the string value using the specified utf 8 json writer.
70          /// </para>
71          /// <para></para>
72          /// </summary>
73          /// <param name="utf8JsonWriter">
74          /// <para>The utf json writer.</para>
75          /// <para></para>
76          /// </param>
77          /// <param name="valueLink">
78          /// <para>The value link.</para>
79          /// <para></para>
80          /// </param>
81          private void WriteStringValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
            ↪ utf8JsonWriter.WriteStringValue(Storage.GetString(valueLink));
82
83          /// <summary>
84          /// <para>
85          /// Writes the string using the specified utf 8 json writer.
86          /// </para>
87          /// <para></para>
88          /// </summary>
89          /// <param name="utf8JsonWriter">
90          /// <para>The utf json writer.</para>
91          /// <para></para>
92          /// </param>
93          /// <param name="parent">
94          /// <para>The parent.</para>
95          /// <para></para>
96          /// </param>
97          /// <param name="valueLink">
98          /// <para>The value link.</para>
99          /// <para></para>
100         /// </param>
101         private void WriteString(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
            ↪ valueLink) => utf8JsonWriter.WriteString(parent, Storage.GetString(valueLink));
102
103         /// <summary>
104         /// <para>
105         /// Writes the number value using the specified utf 8 json writer.
106         /// </para>
107         /// <para></para>
108         /// </summary>
109         /// <param name="utf8JsonWriter">
110         /// <para>The utf json writer.</para>
111         /// <para></para>
112         /// </param>
113         /// <param name="valueLink">
114         /// <para>The value link.</para>
115         /// <para></para>
116         /// </param>
117         private void WriteNumberValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
            ↪ utf8JsonWriter.WriteNumberValue(Storage.GetNumber(valueLink));
118
119         /// <summary>
120         /// <para>
121         /// Writes the number using the specified utf 8 json writer.
122         /// </para>
123         /// <para></para>
124         /// </summary>
125         /// <param name="utf8JsonWriter">
126         /// <para>The utf json writer.</para>
127         /// <para></para>
128         /// </param>
129         /// <param name="parent">
130         /// <para>The parent.</para>
131         /// <para></para>
132         /// </param>
133         /// <param name="valueLink">
134         /// <para>The value link.</para>
135         /// <para></para>
136         /// </param>
137         private void WriteNumber(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
            ↪ valueLink) => utf8JsonWriter.WriteNumber(parent, Storage.GetNumber(valueLink));
138
139         /// <summary>
```

```
140         /// <para>
141         /// Writes the utf 8 json writer.
142         /// </para>
143         /// <para></para>
144         /// </summary>
145         /// <param name="utf8JsonWriter">
146         /// <para>The utf json writer.</para>
147         /// <para></para>
148         /// </param>
149         /// <param name="parent">
150         /// <para>The parent.</para>
151         /// <para></para>
152         /// </param>
153         /// <param name="valueLink">
154         /// <para>The value link.</para>
155         /// <para></para>
156         /// </param>
157         /// <param name="cancellationToken">
158         /// <para>The cancellation token.</para>
159         /// <para></para>
160         /// </param>
161         private void Write(ref Utf8JsonWriter utf8JsonWriter, string parent, TLink valueLink,
              ↪ CancellationToken cancellationToken)
162         {
163             if (cancellationToken.IsCancellationRequested)
164             {
165                 return;
166             }
167             var valueMarker = Storage.GetValueMarker(valueLink);
168             if (EqualityComparer.Equals(valueMarker, Storage.ObjectMarker))
169             {
170                 utf8JsonWriter.WriteStartObject(parent);
171                 var membersLinks = Storage.GetMembersLinks(Storage.GetObject(valueLink));
172                 foreach (var memberLink in membersLinks)
173                 {
174                     if (cancellationToken.IsCancellationRequested)
175                     {
176                         return;
177                     }
178                     Write(ref utf8JsonWriter, Storage.GetString(memberLink),
                          ↪ Storage.GetValueLink(memberLink), cancellationToken);
179                 }
180                 utf8JsonWriter.WriteEndObject();
181             }
182             else if (EqualityComparer.Equals(valueMarker, Storage.ArrayMarker))
183             {
184                 var array = Storage.GetArray(valueLink);
185                 var sequence = Storage.GetArraySequence(array);
186                 utf8JsonWriter.WriteStartArray(parent);
187                 if (!EqualityComparer.Equals(sequence, Storage.EmptyArrayMarker))
188                 {
189                     RightSequenceWalker<TLink> rightSequenceWalker = new(Storage.Links, new
                          ↪ DefaultStack<TLink>(), IsElement);
190                     var elements = rightSequenceWalker.Walk(sequence);
191                     foreach (var element in elements)
192                     {
193                         if (cancellationToken.IsCancellationRequested)
194                         {
195                             return;
196                         }
197                         Write(ref utf8JsonWriter, element, in cancellationToken);
198                     }
199                 }
200                 utf8JsonWriter.WriteEndArray();
201             }
202             else if (EqualityComparer.Equals(valueMarker, Storage.StringMarker))
203             {
204                 WriteString(in utf8JsonWriter, parent, valueLink);
205             }
206             else if (EqualityComparer.Equals(valueMarker, Storage.NumberMarker))
207             {
208                 WriteNumber(in utf8JsonWriter, parent, valueLink);
209             }
210             else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
211             {
212                 utf8JsonWriter.WriteBoolean(parent, true);
213             }
214             else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
```

```
215             {
216                 utf8JsonWriter.WriteBoolean(parent, false);
217             }
218             else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
219             {
220                 utf8JsonWriter.WriteNull(parent);
221             }
222         }
223 
224         /// <summary>
225         /// <para>
226         /// Writes the utf 8 json writer.
227         /// </para>
228         /// <para></para>
229         /// </summary>
230         /// <param name="utf8JsonWriter">
231         /// <para>The utf json writer.</para>
232         /// <para></para>
233         /// </param>
234         /// <param name="valueLink">
235         /// <para>The value link.</para>
236         /// <para></para>
237         /// </param>
238         /// <param name="cancellationToken">
239         /// <para>The cancellation token.</para>
240         /// <para></para>
241         /// </param>
242         private void Write(ref Utf8JsonWriter utf8JsonWriter, TLink valueLink, in
            ↪ CancellationToken cancellationToken)
243         {
244             if (cancellationToken.IsCancellationRequested)
245             {
246                 return;
247             }
248             var valueMarker = Storage.GetValueMarker(valueLink);
249             if (EqualityComparer.Equals(valueMarker, Storage.ObjectMarker))
250             {
251                 utf8JsonWriter.WriteStartObject();
252                 var membersLinks = Storage.GetMembersLinks(Storage.GetObject(valueLink));
253                 foreach (var memberLink in membersLinks)
254                 {
255                     if (cancellationToken.IsCancellationRequested)
256                     {
257                         return;
258                     }
259                     Write(ref utf8JsonWriter, Storage.GetString(memberLink),
                        ↪ Storage.GetValueLink(memberLink), cancellationToken);
260                 }
261                 utf8JsonWriter.WriteEndObject();
262             }
263             else if (EqualityComparer.Equals(valueMarker, Storage.ArrayMarker))
264             {
265                 var array = Storage.GetArray(valueLink);
266                 var sequence = Storage.GetArraySequence(array);
267                 utf8JsonWriter.WriteStartArray();
268                 if (!EqualityComparer.Equals(sequence, Storage.EmptyArrayMarker))
269                 {
270                     RightSequenceWalker<TLink> rightSequenceWalker = new(Storage.Links, new
                        ↪ DefaultStack<TLink>(), IsElement);
271                     var elements = rightSequenceWalker.Walk(sequence);
272                     foreach (var element in elements)
273                     {
274                         if (cancellationToken.IsCancellationRequested)
275                         {
276                             return;
277                         }
278                         Write(ref utf8JsonWriter, element, in cancellationToken);
279                     }
280                 }
281                 utf8JsonWriter.WriteEndArray();
282             }
283             else if (EqualityComparer.Equals(valueMarker, Storage.StringMarker))
284             {
285                 WriteStringValue(in utf8JsonWriter, valueLink);
286             }
287             else if (EqualityComparer.Equals(valueMarker, Storage.NumberMarker))
288             {
289                 WriteNumberValue(in utf8JsonWriter, valueLink);
```

```
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
            {
                utf8JsonWriter.WriteBooleanValue(true);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
            {
                utf8JsonWriter.WriteBooleanValue(false);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
            {
                utf8JsonWriter.WriteNullValue();
            }
        }

        /// <summary>
        /// <para>
        /// Exports the document.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="document">
        /// <para>The document.</para>
        /// <para></para>
        /// </param>
        /// <param name="utf8JsonWriter">
        /// <para>The utf json writer.</para>
        /// <para></para>
        /// </param>
        /// <param name="cancellationToken">
        /// <para>The cancellation token.</para>
        /// <para></para>
        /// </param>
        /// <exception cref="Exception">
        /// <para>No document with this name exists</para>
        /// <para></para>
        /// </exception>
        public void Export(TLink document, ref Utf8JsonWriter utf8JsonWriter, in
        ↪  CancellationToken cancellationToken)
        {
            if (EqualityComparer.Equals(document, default))
            {
                throw new Exception("No document with this name exists");
            }
            var valueLink = Storage.GetValueLink(document);
            Write(ref utf8JsonWriter, valueLink, in cancellationToken);
            utf8JsonWriter.Flush();
        }

        /// <summary>
        /// <para>
        /// Exports the document name.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="documentName">
        /// <para>The document name.</para>
        /// <para></para>
        /// </param>
        /// <param name="utf8JsonWriter">
        /// <para>The utf json writer.</para>
        /// <para></para>
        /// </param>
        /// <param name="cancellationToken">
        /// <para>The cancellation token.</para>
        /// <para></para>
        /// </param>
        public void Export(string documentName, Utf8JsonWriter utf8JsonWriter, CancellationToken
        ↪  cancellationToken) => Export(Storage.GetDocumentOrDefault(documentName), ref
        ↪  utf8JsonWriter, in cancellationToken);
    }
}
```

1.5 ./csharp/Platform.Data.Doublets.Json/JsonExporterCli.cs

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

```
using Platform.IO;
using System.Text.Json;
using Platform.Data.Doublets.Memory;
using Platform.Data.Doublets.Sequences.Converters;
using Platform.Memory;

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

namespace Platform.Data.Doublets.Json
{
    /// <summary>
    /// <para>
    /// Represents the json exporter cli.
    /// </para>
    /// <para></para>
    /// </summary>
    public class JsonExporterCli<TLink>
        where TLink : struct
    {
        /// <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 argumentIndex = 0;
            var linksFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Links file path", args);
            var jsonFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "JSON file path", args);
            var defaultDocumentName = Path.GetFileNameWithoutExtension(jsonFilePath);
            var documentName = ConsoleHelpers.GetOrReadArgument(argumentIndex, $"Document name (default: {defaultDocumentName})", args);
            if (string.IsNullOrWhiteSpace(documentName))
            {
                documentName = defaultDocumentName;
            }
            if (!File.Exists(linksFilePath))
            {
                Console.WriteLine($"${linksFilePath} file does not exist.");
            }
            using FileStream jsonFileStream = new(jsonFilePath, FileMode.Append);
            JsonWriterOptions utf8JsonWriterOptions = new()
            {
                Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping,
                Indented = true
            };
            Utf8JsonWriter utf8JsonWriter = new(jsonFileStream, utf8JsonWriterOptions);
            var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport: true);
            using UnitedMemoryLinks<TLink> memoryAdapter = new (new FileMappedResizableDirectMemory(linksFilePath), UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants, IndexTreeType.Default);
            var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
            BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
            var storage = new DefaultJsonStorage<TLink>(links, balancedVariantConverter);
            var exporter = new JsonExporter<TLink>(storage);
            var document = storage.GetDocumentOrDefault(documentName);
            if (storage.EqualityComparer.Equals(document, default))
            {
                Console.WriteLine("No document with this name.");
            }
            using ConsoleCancellation cancellation = new ();
            var cancellationToken = cancellation.Token;
            Console.WriteLine("Press CTRL+C to stop.");
            try
            {
                exporter.Export(document, ref utf8JsonWriter, in cancellationToken);
            }
            catch (Exception exception)
            {
```

```
                Console.WriteLine(exception);
                return;
            }
            finally
            {
                utf8JsonWriter.Dispose();
            }
            Console.WriteLine("Export completed successfully.");
        }
    }
}
```

1.6 ./csharp/Platform.Data.Doublets.Json/JsonImporter.cs

```
using System;
using System.Collections.Generic;
using System.Text.Json;
using System.Threading;

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

namespace Platform.Data.Doublets.Json
{
    /// <summary>
    /// <para>
    /// Represents the json importer.
    /// </para>
    /// <para></para>
    /// </summary>
    public class JsonImporter<TLink>
    {
        /// <summary>
        /// <para>
        /// The storage.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly IJsonStorage<TLink> Storage;
        /// <summary>
        /// <para>
        /// The default.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly EqualityComparer<TLink> EqualityComparer =
        ↪  EqualityComparer<TLink>.Default;
        /// <summary>
        /// <para>
        /// The parents.
        /// </para>
        /// <para></para>
        /// </summary>
        public readonly Stack<TLink> Parents = new ();
        /// <summary>
        /// <para>
        /// Initializes a new <see cref="JsonImporter"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="storage">
        /// <para>A storage.</para>
        /// <para></para>
        /// </param>
        public JsonImporter(IJsonStorage<TLink> storage) => Storage = storage;

            /// <summary>
            /// <para>
            /// Pops the if parent is member.
            /// </para>
            /// <para></para>
            /// </summary>
            private void PopIfParentIsMember()
        {
            var parent = Parents.Peek();
            var parentMarker = Storage.GetValueMarker(parent);
            if (EqualityComparer.Equals(parentMarker, Storage.MemberMarker))
            {
                Parents.Pop();
            }
        }
```

```
66
67          /// <summary>
68          /// <para>
69          /// Imports the document name.
70          /// </para>
71          /// <para></para>
72          /// </summary>
73          /// <param name="documentName">
74          /// <para>The document name.</para>
75          /// <para></para>
76          /// </param>
77          /// <param name="utf8JsonReader">
78          /// <para>The utf json reader.</para>
79          /// <para></para>
80          /// </param>
81          /// <param name="cancellationToken">
82          /// <para>The cancellation token.</para>
83          /// <para></para>
84          /// </param>
85          /// <exception cref="Exception">
86          /// <para>The document with the specified name already exists.</para>
87          /// <para></para>
88          /// </exception>
89          /// <returns>
90          /// <para>The document.</para>
91          /// <para></para>
92          /// </returns>
93          public TLink Import(string documentName, ref Utf8JsonReader utf8JsonReader, in
            ↪ CancellationToken cancellationToken)
94          {
95              Parents.Clear();
96              if (!EqualityComparer.Equals(Storage.GetDocumentOrDefault(documentName), default))
97              {
98                  throw new Exception("The document with the specified name already exists.");
99              }
100             var document = Storage.CreateDocument(documentName);
101             Parents.Push(document);
102             TLink parent;
103             TLink parentMarker;
104             JsonTokenType tokenType;
105             TLink value;
106             TLink newParentArray;
107             while (utf8JsonReader.Read())
108             {
109                 cancellationToken.ThrowIfCancellationRequested();
110                 parent = Parents.Peek();
111                 parentMarker = Storage.GetValueMarker(parent);
112                 tokenType = utf8JsonReader.TokenType;
113                 if (utf8JsonReader.TokenType == JsonTokenType.PropertyName)
114                 {
115                     var @object = Storage.GetObject(parent);
116                     var property = utf8JsonReader.GetString();
117                     Parents.Push(Storage.AttachMemberToObject(@object, property));
118                 }
119                 switch (tokenType)
120                 {
121                     case JsonTokenType.StartObject:
122                     {
123                         value = Storage.CreateObjectValue();
124                         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
125                         {
126                             Parents.Pop();
127                             newParentArray = Storage.AppendArrayValue(parent, value);
128                             Parents.Push(newParentArray);
129                             Parents.Push(value);
130                         }
131                         else
132                         {
133                             var @object = Storage.Attach(parent, value);
134                             Parents.Push(@object);
135                         }
136                         break;
137                     }
138                     case JsonTokenType.EndObject:
139                         Parents.Pop();
140                         break;
141                     case JsonTokenType.StartArray:
142                         value = Storage.CreateArrayValue(Array.Empty<TLink>());
```

```
                    Parents.Push(value);
                    break;
                case JsonTokenType.EndArray:
                {
                    var arrayValue = Parents.Pop();
                    parent = Parents.Peek();
                    parentMarker = Storage.GetValueMarker(parent);
                    if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                    {
                        Parents.Pop();
                        newParentArray = Storage.AppendArrayValue(parent, arrayValue);
                        Parents.Push(newParentArray);
                    }
                    Storage.Attach(parent, arrayValue);
                    break;
                }
                case JsonTokenType.String:
                {
                    var @string = utf8JsonReader.GetString();
                    value = Storage.CreateStringValue(@string);
                    if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                    {
                        Parents.Pop();
                        newParentArray = Storage.AppendArrayValue(parent, value);
                        Parents.Push(newParentArray);
                    }
                    else
                    {
                        Storage.Attach(parent, value);
                    }
                    break;
                }
                case JsonTokenType.Number:
                {
                    value = Storage.CreateNumberValue(utf8JsonReader.GetDecimal());
                    if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                    {
                        Parents.Pop();
                        newParentArray = Storage.AppendArrayValue(parent, value);
                        Parents.Push(newParentArray);
                    }
                    else
                    {
                        Storage.Attach(parent, value);
                    }
                    break;
                }
                case JsonTokenType.True:
                {
                    value = Storage.CreateBooleanValue(true);
                    if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                    {
                        Parents.Pop();
                        newParentArray = Storage.AppendArrayValue(parent, value);
                        Parents.Push(newParentArray);
                    }
                    else
                    {
                        Storage.Attach(parent, value);
                    }
                    break;
                }
                case JsonTokenType.False:
                {
                    value = Storage.CreateBooleanValue(false);
                    if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                    {
                        Parents.Pop();
                        newParentArray = Storage.AppendArrayValue(parent, value);
                        Parents.Push(newParentArray);
                    }
                    else
                    {
                        Storage.Attach(parent, value);
                    }
                    break;
                }
                case JsonTokenType.Null:
                {
```

```
                        value = Storage.CreateNullValue();
                        if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                        {
                            Parents.Pop();
                            newParentArray = Storage.AppendArrayValue(parent, value);
                            Parents.Push(newParentArray);
                        }
                        else
                        {
                            Storage.Attach(parent, value);
                        }
                        break;
                    }
                }
                if (tokenType != JsonTokenType.PropertyName && tokenType !=
                ↪  JsonTokenType.StartObject && tokenType != JsonTokenType.StartArray)
                {
                    PopIfParentIsMember();
                }
            }
            return document;
        }
    }
}
```

1.7 ./csharp/Platform.Data.Doublets.Json/JsonImporterCli.cs

```
using System;
using System.IO;
using System.Text;
using Platform.Data.Doublets.Memory.United.Generic;
using Platform.IO;
using System.Text.Json;
using Platform.Data.Doublets.Memory;
using Platform.Data.Doublets.Sequences.Converters;
using Platform.Memory;

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

namespace Platform.Data.Doublets.Json
{
    /// <summary>
    /// <para>
    /// Represents the json importer cli.
    /// </para>
    /// <para></para>
    /// </summary>
    public class JsonImporterCli<TLink>
        where TLink : struct
    {
        /// <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 argumentIndex = 0;
            var jsonFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "JSON file
            ↪  path", args);
            var linksFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Links file
            ↪  path", args);
            var defaultDocumentName = Path.GetFileNameWithoutExtension(jsonFilePath);
            var documentName = ConsoleHelpers.GetOrReadArgument(argumentIndex, $"Document name
            ↪  (default: {defaultDocumentName})", args);
            if (string.IsNullOrWhiteSpace(documentName))
            {
                documentName = defaultDocumentName;
            }
            if (!File.Exists(jsonFilePath))
            {
                Console.WriteLine($"${jsonFilePath} file does not exist.");
            }
            var json = File.ReadAllText(jsonFilePath);
            var encodedJson = Encoding.UTF8.GetBytes(json);
```

```
            ReadOnlySpan<byte> readOnlySpanEncodedJson = new(encodedJson);
            Utf8JsonReader utf8JsonReader = new(readOnlySpanEncodedJson);
            LinksConstants<TLink> linksConstants = new(enableExternalReferencesSupport: true);
            FileMappedResizableDirectMemory fileMappedResizableDirectMemory = new(linksFilePath);
            var unitedMemoryLinks = UnitedMemoryLinks<TLink>.DefaultLinksSizeStep;
            const IndexTreeType indexTreeType = IndexTreeType.Default;
            using UnitedMemoryLinks<TLink> memoryAdapter = new(fileMappedResizableDirectMemory,
            ↪  unitedMemoryLinks, linksConstants, indexTreeType);
            var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
            BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
            DefaultJsonStorage<TLink> storage = new(links, balancedVariantConverter);
            JsonImporter<TLink> importer = new(storage);
            using ConsoleCancellation cancellation = new();
            var cancellationToken = cancellation.Token;
            Console.WriteLine("Press CTRL+C to stop.");
            try
            {
                importer.Import(documentName, ref utf8JsonReader, in cancellationToken);
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception);
                return;
            }
            Console.WriteLine("Import completed successfully.");
        }
    }
}
```

1.8 ./csharp/Platform.Data.Doublets.Json.Tests/JsonImportAndExportTests.cs

```
using System.Text;
using System.Text.Json;
using System.Threading;
using System.IO;
using Xunit;
using TLink = System.UInt64;
using Platform.Data.Doublets.Memory.United.Generic;
using Platform.Memory;
using Platform.Data.Doublets.Memory;
using System.Text.RegularExpressions;
using Platform.Data.Doublets.Sequences.Converters;

namespace Platform.Data.Doublets.Json.Tests
{
    /// <summary>
    /// <para>
    /// Represents the json import and export tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class JsonImportAndExportTests
    {
        /// <summary>
        /// <para>
        /// The balanced variant converter.
        /// </para>
        /// <para></para>
        /// </summary>
        public static BalancedVariantConverter<TLink> BalancedVariantConverter;

        /// <summary>
        /// <para>
        /// Creates the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A links of t link</para>
        /// <para></para>
        /// </returns>
        public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new IO.TemporaryFile());

        /// <summary>
        /// <para>
        /// Creates the links using the specified data db filename.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
```

```
50          /// <para>The link.</para>
51          /// <para></para>
52          /// </typeparam>
53          /// <param name="dataDBFilename">
54          /// <para>The data db filename.</para>
55          /// <para></para>
56          /// </param>
57          /// <returns>
58          /// <para>A links of t link</para>
59          /// <para></para>
60          /// </returns>
61          public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
62          {
63              var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
                ↪  true);
64              return new UnitedMemoryLinks<TLink>(new
                ↪  FileMappedResizableDirectMemory(dataDBFilename),
                ↪  UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
                ↪  IndexTreeType.Default);
65          }
66
67          /// <summary>
68          /// <para>
69          /// Creates the json storage using the specified links.
70          /// </para>
71          /// <para></para>
72          /// </summary>
73          /// <param name="links">
74          /// <para>The links.</para>
75          /// <para></para>
76          /// </param>
77          /// <returns>
78          /// <para>A default json storage of t link</para>
79          /// <para></para>
80          /// </returns>
81          public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links) => new
                ↪  (links, BalancedVariantConverter);
82
83          /// <summary>
84          /// <para>
85          /// Imports the storage.
86          /// </para>
87          /// <para></para>
88          /// </summary>
89          /// <param name="storage">
90          /// <para>The storage.</para>
91          /// <para></para>
92          /// </param>
93          /// <param name="documentName">
94          /// <para>The document name.</para>
95          /// <para></para>
96          /// </param>
97          /// <param name="json">
98          /// <para>The json.</para>
99          /// <para></para>
100         /// </param>
101         /// <returns>
102         /// <para>The link</para>
103         /// <para></para>
104         /// </returns>
105         public TLink Import(IJsonStorage<TLink> storage, string documentName, byte[] json)
106         {
107             Utf8JsonReader utf8JsonReader = new(json);
108             JsonImporter<TLink> jsonImporter = new(storage);
109             CancellationTokenSource importCancellationTokenSource = new();
110             CancellationToken cancellationToken = importCancellationTokenSource.Token;
111             return jsonImporter.Import(documentName, ref utf8JsonReader, in cancellationToken);
112         }
113
114         /// <summary>
115         /// <para>
116         /// Exports the document link.
117         /// </para>
118         /// <para></para>
119         /// </summary>
120         /// <param name="documentLink">
121         /// <para>The document link.</para>
122         /// <para></para>
```

```
        /// </param>
        /// <param name="storage">
        /// <para>The storage.</para>
        /// <para></para>
        /// </param>
        /// <param name="stream">
        /// <para>The stream.</para>
        /// <para></para>
        /// </param>
        public void Export(TLink documentLink, IJsonStorage<TLink> storage, in MemoryStream
        ↪  stream)
        {
            Utf8JsonWriter writer = new(stream);
            JsonExporter<TLink> jsonExporter = new(storage);
            CancellationTokenSource exportCancellationTokenSource = new();
            CancellationToken exportCancellationToken = exportCancellationTokenSource.Token;
            jsonExporter.Export(documentLink, ref writer, in exportCancellationToken);
            writer.Dispose();
        }

        /// <summary>
        /// <para>
        /// Tests that test.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="initialJson">
        /// <para>The initial json.</para>
        /// <para></para>
        /// </param>
        [Theory]
        [InlineData("{}")]
        [InlineData("\"stringValue\"")]
        [InlineData("228")]
        [InlineData("0.5")]
        [InlineData("[]")]
        [InlineData("true")]
        [InlineData("false")]
        [InlineData("null")]
        [InlineData("{ \"string\": \"string\" }")]
        [InlineData("{ \"null\": null }")]
        [InlineData("{ \"boolean\": false }")]
        [InlineData("{ \"boolean\": true }")]
        [InlineData("{ \"array\": [] }")]
        [InlineData("{ \"array\": [1] }")]
        [InlineData("{ \"object\": {} }")]
        [InlineData("{ \"number\": 1 }")]
        [InlineData("{ \"decimal\": 0.5 }")]
        [InlineData("[null]")]
        [InlineData("[true]")]
        [InlineData("[false]")]
        [InlineData("[[]]")]
        [InlineData("[[1]]")]
        [InlineData("[[0.5]]")]
        [InlineData("[{}]")]
        [InlineData("[\"The Venus Project\"]")]
        [InlineData("[{ \"title\": \"The Venus Project\" }]")]
        [InlineData("[1,2,3,4]")]
        [InlineData("[-0.5, 0.5]")]
        public void Test(string initialJson)
        {
            var links = CreateLinks();
            BalancedVariantConverter = new(links);
            var storage = CreateJsonStorage(links);
            var json = Encoding.UTF8.GetBytes(initialJson);
            var documentLink = Import(storage, "documentName", json);
            MemoryStream stream = new();
            Export(documentLink, storage, in stream);
            string exportedJson = Encoding.UTF8.GetString(stream.ToArray());
            stream.Dispose();
            var minimizedInitialJson = Regex.Replace(initialJson,
            ↪  "(\"(?:[^\"\\\\]|\\\\.)*\")|\\s+", "$1");
            Assert.Equal(minimizedInitialJson, exportedJson);
        }
    }
}
```

1.9 ./csharp/Platform.Data.Doublets.Json.Tests/JsonStorageTests.cs

```
using Xunit;
using Platform.Data.Doublets.Memory.United.Generic;
using Platform.Data.Doublets.Memory;
using Platform.Memory;
using TLink = System.UInt32;
using Xunit.Abstractions;
using Platform.Collections.Stacks;
using Platform.Data.Doublets.Sequences.Walkers;
using System.Collections.Generic;
using Platform.Data.Doublets.Sequences.Converters;

namespace Platform.Data.Doublets.Json.Tests
{
    /// <summary>
    /// <para>
    /// Represents the json storage tests.
    /// </para>
    /// <para></para>
    /// </summary>
    public class JsonStorageTests
    {
        /// <summary>
        /// <para>
        /// The output.
        /// </para>
        /// <para></para>
        /// </summary>
        private readonly ITestOutputHelper output;
        /// <summary>
        /// <para>
        /// The balanced variant converter.
        /// </para>
        /// <para></para>
        /// </summary>
        public static BalancedVariantConverter<TLink> BalancedVariantConverter;

        /// <summary>
        /// <para>
        /// Initializes a new <see cref="JsonStorageTests"/> instance.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <param name="output">
        /// <para>A output.</para>
        /// <para></para>
        /// </param>
        public JsonStorageTests(ITestOutputHelper output)
        {
            this.output = output;
        }

        /// <summary>
        /// <para>
        /// Creates the links.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <returns>
        /// <para>A links of t link</para>
        /// <para></para>
        /// </returns>
        public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
        ↪  Platform.IO.TemporaryFile());

        /// <summary>
        /// <para>
        /// Creates the links using the specified data db filename.
        /// </para>
        /// <para></para>
        /// </summary>
        /// <typeparam name="TLink">
        /// <para>The link.</para>
        /// <para></para>
        /// </typeparam>
        /// <param name="dataDBFilename">
        /// <para>The data db filename.</para>
        /// <para></para>
        /// </param>
```

```
78          /// <returns>
79          /// <para>A links of t link</para>
80          /// <para></para>
81          /// </returns>
82          public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
83          {
84              var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
                ↪  true);
85              return new UnitedMemoryLinks<TLink>(new
                ↪  FileMappedResizableDirectMemory(dataDBFilename),
                ↪  UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
                ↪  IndexTreeType.Default);
86          }
87
88          /// <summary>
89          /// <para>
90          /// Creates the json storage.
91          /// </para>
92          /// <para></para>
93          /// </summary>
94          /// <returns>
95          /// <para>A default json storage of t link</para>
96          /// <para></para>
97          /// </returns>
98          public static DefaultJsonStorage<TLink> CreateJsonStorage()
99          {
100             var links = CreateLinks();
101             return CreateJsonStorage(links);
102         }
103
104         /// <summary>
105         /// <para>
106         /// Creates the json storage using the specified links.
107         /// </para>
108         /// <para></para>
109         /// </summary>
110         /// <param name="links">
111         /// <para>The links.</para>
112         /// <para></para>
113         /// </param>
114         /// <returns>
115         /// <para>A default json storage of t link</para>
116         /// <para></para>
117         /// </returns>
118         public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links)
119         {
120             BalancedVariantConverter = new(links);
121             return new DefaultJsonStorage<TLink>(links, BalancedVariantConverter);
122         }
123
124         /// <summary>
125         /// <para>
126         /// Tests that constructors test.
127         /// </para>
128         /// <para></para>
129         /// </summary>
130         [Fact]
131         public void ConstructorsTest() => CreateJsonStorage();
132
133         /// <summary>
134         /// <para>
135         /// Tests that create document test.
136         /// </para>
137         /// <para></para>
138         /// </summary>
139         [Fact]
140         public void CreateDocumentTest()
141         {
142             var defaultJsonStorage = CreateJsonStorage();
143             defaultJsonStorage.CreateDocument("documentName");
144         }
145
146         /// <summary>
147         /// <para>
148         /// Tests that get document test.
149         /// </para>
150         /// <para></para>
151         /// </summary>
```

```
        [Fact]
        public void GetDocumentTest()
        {
            var defaultJsonStorage = CreateJsonStorage();
            var createdDocumentLink = defaultJsonStorage.CreateDocument("documentName");
            var foundDocumentLink = defaultJsonStorage.GetDocumentOrDefault("documentName");
            Assert.Equal(createdDocumentLink, foundDocumentLink);
        }

        /// <summary>
        /// <para>
        /// Tests that create object test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void CreateObjectTest()
        {
            var defaultJsonStorage = CreateJsonStorage();
            var object0 = defaultJsonStorage.CreateObjectValue();
            var object1 = defaultJsonStorage.CreateObjectValue();
            Assert.NotEqual(object0, object1);
        }

        /// <summary>
        /// <para>
        /// Tests that create string test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void CreateStringTest()
        {
            var defaultJsonStorage = CreateJsonStorage();
            defaultJsonStorage.CreateString("string");
        }

        /// <summary>
        /// <para>
        /// Tests that create member test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void CreateMemberTest()
        {
            var defaultJsonStorage = CreateJsonStorage();
            var document = defaultJsonStorage.CreateDocument("documentName");
            defaultJsonStorage.AttachObject(document);
            defaultJsonStorage.CreateMember("keyName");
        }

        /// <summary>
        /// <para>
        /// Tests that attach object value to document test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachObjectValueToDocumentTest()
        {
            var links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentValueLink = defaultJsonStorage.AttachObject(document);
            TLink createdObjectValue = links.GetTarget(documentValueLink);

            TLink valueMarker = links.GetSource(createdObjectValue);
            Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);

            TLink createdObject = links.GetTarget(createdObjectValue);
            TLink objectMarker = links.GetSource(createdObject);
            Assert.Equal(objectMarker, defaultJsonStorage.ObjectMarker);

            TLink foundDocumentValue = defaultJsonStorage.GetValueLink(document);
            Assert.Equal(createdObjectValue, foundDocumentValue);
        }

```

```
        /// <summary>
        /// <para>
        /// Tests that attach string value to document test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachStringValueToDocumentTest()
        {
            var links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentStringLink = defaultJsonStorage.AttachString(document, "stringName");
            TLink createdStringValue = links.GetTarget(documentStringLink);

            TLink valueMarker = links.GetSource(createdStringValue);
            Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);

            TLink createdString = links.GetTarget(createdStringValue);
            TLink stringMarker = links.GetSource(createdString);
            Assert.Equal(stringMarker, defaultJsonStorage.StringMarker);

            TLink foundStringValue = defaultJsonStorage.GetValueLink(document);
            Assert.Equal(createdStringValue, foundStringValue);
        }

        /// <summary>
        /// <para>
        /// Tests that attach number to document test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachNumberToDocumentTest()
        {
            var links = CreateLinks();
            var defaultJsonStorage = CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentNumberLink = defaultJsonStorage.AttachNumber(document, 2021);
            TLink createdNumberValue = links.GetTarget(documentNumberLink);

            TLink valueMarker = links.GetSource(createdNumberValue);
            Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);

            TLink createdNumber = links.GetTarget(createdNumberValue);
            TLink numberMarker = links.GetSource(createdNumber);
            Assert.Equal(numberMarker, defaultJsonStorage.NumberMarker);

            TLink foundNumberValue = defaultJsonStorage.GetValueLink(document);
            Assert.Equal(createdNumberValue, foundNumberValue);
        }

        /// <summary>
        /// <para>
        /// Tests that attach true value to document test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachTrueValueToDocumentTest()
        {
            var links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");

            TLink documentTrueValueLink = defaultJsonStorage.AttachBoolean(document, true);
            TLink createdTrueValue = links.GetTarget(documentTrueValueLink);

            TLink valueMarker = links.GetSource(createdTrueValue);
            Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);

            TLink trueMarker = links.GetTarget(createdTrueValue);
            Assert.Equal(trueMarker, defaultJsonStorage.TrueMarker);

            TLink foundTrueValue = defaultJsonStorage.GetValueLink(document);
            Assert.Equal(createdTrueValue, foundTrueValue);
        }

        /// <summary>
```

```
        /// <para>
        /// Tests that attach false value to document test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachFalseValueToDocumentTest()
        {
            var links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");

            TLink documentFalseValueLink = defaultJsonStorage.AttachBoolean(document, false);
            TLink createdFalseValue = links.GetTarget(documentFalseValueLink);

            TLink valueMarker = links.GetSource(createdFalseValue);
            Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);

            TLink falseMarker = links.GetTarget(createdFalseValue);
            Assert.Equal(falseMarker, defaultJsonStorage.FalseMarker);

            TLink foundFalseValue = defaultJsonStorage.GetValueLink(document);
            Assert.Equal(createdFalseValue, foundFalseValue);
        }

        /// <summary>
        /// <para>
        /// Tests that attach null value to document test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachNullValueToDocumentTest()
        {
            var links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");

            TLink documentNullValueLink = defaultJsonStorage.AttachNull(document);
            TLink createdNullValue = links.GetTarget(documentNullValueLink);

            TLink valueMarker = links.GetSource(createdNullValue);
            Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);

            TLink nullMarker = links.GetTarget(createdNullValue);
            Assert.Equal(nullMarker, defaultJsonStorage.NullMarker);

            TLink foundNullValue = defaultJsonStorage.GetValueLink(document);
            Assert.Equal(createdNullValue, foundNullValue);
        }

        /// <summary>
        /// <para>
        /// Tests that attach empty array value to document test.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachEmptyArrayValueToDocumentTest()
        {
            var links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");

            TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, new
            ↪  TLink[0]);
            TLink createdArrayValue = links.GetTarget(documentArrayValueLink);
            output.WriteLine(links.Format(createdArrayValue));


            TLink valueMarker = links.GetSource(createdArrayValue);
            Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);

            TLink createdArrayLink = links.GetTarget(createdArrayValue);
            TLink arrayMarker = links.GetSource(createdArrayLink);
            Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);

            TLink createArrayContents = links.GetTarget(createdArrayLink);
            Assert.Equal(createArrayContents, defaultJsonStorage.EmptyArrayMarker);
```

```
387
388             TLink foundArrayValue = defaultJsonStorage.GetValueLink(document);
389             Assert.Equal(createdArrayValue, foundArrayValue);
390         }
391
392         /// <summary>
393         /// <para>
394         /// Tests that attach array value to document test.
395         /// </para>
396         /// <para></para>
397         /// </summary>
398         [Fact]
399         public void AttachArrayValueToDocumentTest()
400         {
401             var links = CreateLinks();
402             var defaultJsonStorage =CreateJsonStorage(links);
403             TLink document = defaultJsonStorage.CreateDocument("documentName");
404
405             TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
406             TLink[] array = new TLink[] { arrayElement, arrayElement, arrayElement };
407
408
409             TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, array);
410             TLink createdArrayValue = links.GetTarget(documentArrayValueLink);
411
412             DefaultStack<TLink> stack = new();
413             RightSequenceWalker<TLink> rightSequenceWalker = new(links, stack, arrayElementLink
                ↪  => links.GetSource(arrayElementLink) == defaultJsonStorage.ValueMarker);
414             IEnumerable<TLink> arrayElementsValuesLink =
                ↪  rightSequenceWalker.Walk(createdArrayValue);
415             Assert.NotEmpty(arrayElementsValuesLink);
416
417             output.WriteLine(links.Format(createdArrayValue));
418
419
420             TLink valueMarker = links.GetSource(createdArrayValue);
421             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
422
423             TLink createdArrayLink = links.GetTarget(createdArrayValue);
424             TLink arrayMarker = links.GetSource(createdArrayLink);
425             Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
426
427             TLink createdArrayContents = links.GetTarget(createdArrayLink);
428             Assert.Equal(links.GetTarget(createdArrayContents), arrayElement);
429
430
431             TLink foundArrayValue = defaultJsonStorage.GetValueLink(document);
432             Assert.Equal(createdArrayValue, foundArrayValue);
433         }
434
435         /// <summary>
436         /// <para>
437         /// Tests that get object from document object value link test.
438         /// </para>
439         /// <para></para>
440         /// </summary>
441         [Fact]
442         public void GetObjectFromDocumentObjectValueLinkTest()
443         {
444             ILinks<TLink> links = CreateLinks();
445             var defaultJsonStorage =CreateJsonStorage(links);
446             TLink document = defaultJsonStorage.CreateDocument("documentName");
447             TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
448             TLink objectValueLink = links.GetTarget(documentObjectValueLink);
449             TLink objectFromGetObject = defaultJsonStorage.GetObject(documentObjectValueLink);
450             output.WriteLine(links.Format(objectValueLink));
451             output.WriteLine(links.Format(objectFromGetObject));
452             Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
453         }
454
455         /// <summary>
456         /// <para>
457         /// Tests that get object from object value link test.
458         /// </para>
459         /// <para></para>
460         /// </summary>
461         [Fact]
462         public void GetObjectFromObjectValueLinkTest()
463         {
```

```
            ILinks<TLink> links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
            TLink objectValueLink = links.GetTarget(documentObjectValueLink);
            TLink objectFromGetObject = defaultJsonStorage.GetObject(objectValueLink);
            Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
        }

        /// <summary>
        /// <para>
        /// Tests that attach string value to key.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachStringValueToKey()
        {
            ILinks<TLink> links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
            TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
            TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
            TLink memberStringValueLink = defaultJsonStorage.AttachString(memberLink,
            ↪  "stringValue");
            TLink stringValueLink = links.GetTarget(memberStringValueLink);
            List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
            Assert.Equal(memberLink, objectMembersLinks[0]);
            Assert.Equal(stringValueLink,
            ↪  defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
        }

        /// <summary>
        /// <para>
        /// Tests that attach number value to key.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachNumberValueToKey()
        {
            ILinks<TLink> links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
            TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
            TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
            TLink memberNumberValueLink = defaultJsonStorage.AttachNumber(memberLink, 123);
            TLink numberValueLink = links.GetTarget(memberNumberValueLink);
            List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
            Assert.Equal(memberLink, objectMembersLinks[0]);
            Assert.Equal(numberValueLink,
            ↪  defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
        }

        /// <summary>
        /// <para>
        /// Tests that attach object value to key.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachObjectValueToKey()
        {
            ILinks<TLink> links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
            TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
            TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
            TLink memberObjectValueLink = defaultJsonStorage.AttachObject(memberLink);
            TLink objectValueLink = links.GetTarget(memberObjectValueLink);
            List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
            Assert.Equal(memberLink, objectMembersLinks[0]);
            Assert.Equal(objectValueLink,
            ↪  defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
```

```
        }

        /// <summary>
        /// <para>
        /// Tests that attach array value to key.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachArrayValueToKey()
        {
            ILinks<TLink> links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
            TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
            TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
            TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
            TLink[] array = { arrayElement, arrayElement, arrayElement };
            TLink memberArrayValueLink = defaultJsonStorage.AttachArray(memberLink, array);
            TLink arrayValueLink = links.GetTarget(memberArrayValueLink);
            List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
            Assert.Equal(memberLink, objectMembersLinks[0]);
            Assert.Equal(arrayValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
        }

        /// <summary>
        /// <para>
        /// Tests that attach true value to key.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachTrueValueToKey()
        {
            ILinks<TLink> links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
            TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
            TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
            TLink memberTrueValueLink = defaultJsonStorage.AttachBoolean(memberLink, true);
            TLink trueValueLink = links.GetTarget(memberTrueValueLink);
            List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
            Assert.Equal(memberLink, objectMembersLinks[0]);
            Assert.Equal(trueValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
        }

        /// <summary>
        /// <para>
        /// Tests that attach false value to key.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachFalseValueToKey()
        {
            ILinks<TLink> links = CreateLinks();
            var defaultJsonStorage =CreateJsonStorage(links);
            TLink document = defaultJsonStorage.CreateDocument("documentName");
            TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
            TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
            TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
            TLink memberFalseValueLink = defaultJsonStorage.AttachBoolean(memberLink, false);
            TLink falseValueLink = links.GetTarget(memberFalseValueLink);
            List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
            Assert.Equal(memberLink, objectMembersLinks[0]);
            Assert.Equal(falseValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
        }

        /// <summary>
        /// <para>
        /// Tests that attach null value to key.
        /// </para>
        /// <para></para>
        /// </summary>
        [Fact]
        public void AttachNullValueToKey()
```

```
615         {
616             ILinks<TLink> links = CreateLinks();
617             var defaultJsonStorage =CreateJsonStorage(links);
618             TLink document = defaultJsonStorage.CreateDocument("documentName");
619             TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
620             TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
621             TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
622             TLink memberNullValueLink = defaultJsonStorage.AttachNull(memberLink);
623             TLink nullValueLink = links.GetTarget(memberNullValueLink);
624             List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
625             Assert.Equal(nullValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
626         }
627     }
628 }
```

Index