

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

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

```
1  using Platform.Numbers;
2  using Platform.Data.Doublets.Unicode;
3  using Platform.Data.Doublets.Sequences.Converters;
4  using Platform.Data.Doublets.CriterionMatchers;
5  using Platform.Data.Numbers.Raw;
6  using Platform.Converters;
7  using Platform.Data.Doublets.Sequences.Walkers;
8  using Platform.Collections.Stacks;
9  using System;
10 using System.Collections.Generic;
11 using Platform.Data.Doublets.Numbers.Rational;
12 using Platform.Data.Doublets.Numbers.Raw;
13 using Platform.Data.Doublets.Sequences.HeightProviders;
14 using Platform.Data.Doublets.Sequences;
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 default json storage.
23     /// </para>
24     /// <para></para>
25     /// </summary>
26     /// <seealso cref="IJsonStorage{TLink}" />
27     public class DefaultJsonStorage<TLink> : IJsonStorage<TLink>
28     where TLink : struct
29     {
30         /// <summary>
31         /// <para>
32         /// The any.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         public readonly TLink Any;
37         /// <summary>
38         /// <para>
39         /// The zero.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         public static readonly TLink Zero = default;
44         /// <summary>
45         /// <para>
46         /// The zero.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         public static readonly TLink One = Arithmetic.Increment(Zero);
51         /// <summary>
52         /// <para>
53         /// The balanced variant converter.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         public readonly BalancedVariantConverter<TLink> BalancedVariantConverter;
58         /// <summary>
59         /// <para>
60         /// The list to sequence converter.
61         /// </para>
62         /// <para></para>
63         /// </summary>
64         public readonly IConverter<IList<TLink>, TLink> ListToSequenceConverter;
65         /// <summary>
66         /// <para>
67         /// The meaning root.
68         /// </para>
69         /// <para></para>
70         /// </summary>
71         public readonly TLink MeaningRoot;
72         /// <summary>
73         /// <para>
74         /// The default.
75         /// </para>
76         /// <para></para>
77         /// </summary>
```

```

78 public readonly EqualityComparer<TLink> EqualityComparer =
79     ↳ EqualityComparer<TLink>.Default;
80 // Converters that are able to convert link's address (UInt64 value) to a raw number
81     ↳ represented with another UInt64 value and back
82 /// <summary>
83 /// <para>
84 /// The number to address converter.
85 /// </para>
86 /// <para></para>
87 /// </summary>
88 public readonly RawNumberToAddressConverter<TLink> NumberToAddressConverter = new();
89 /// <summary>
90 /// <para>
91 /// The address to number converter.
92 /// </para>
93 /// <para></para>
94 /// </summary>
95 public readonly AddressToRawNumberConverter<TLink> AddressToNumberConverter = new();
96 // Converters between BigInteger and raw number sequence
97 /// <summary>
98 /// <para>
99 /// The big integer to raw number sequence converter.
100 /// </para>
101 /// <para></para>
102 /// </summary>
103 public readonly BigIntegerToRawNumberSequenceConverter<TLink>
104     ↳ BigIntegerToRawNumberSequenceConverter;
105 /// <summary>
106 /// <para>
107 /// The raw number sequence to big integer converter.
108 /// </para>
109 /// <para></para>
110 /// </summary>
111 public readonly RawNumberSequenceToBigIntegerConverter<TLink>
112     ↳ RawNumberSequenceToBigIntegerConverter;
113 // Converters between decimal and rational number sequence
114 /// <summary>
115 /// <para>
116 /// The decimal to rational converter.
117 /// </para>
118 /// <para></para>
119 /// </summary>
120 public readonly DecimalToRationalConverter<TLink> DecimalToRationalConverter;
121 /// <summary>
122 /// <para>
123 /// The rational to decimal converter.
124 /// </para>
125 /// <para></para>
126 /// </summary>
127 public readonly RationalToDecimalConverter<TLink> RationalToDecimalConverter;
128 // Converters between string and unicode sequence
129 /// <summary>
130 /// <para>
131 /// The string to unicode sequence converter.
132 /// </para>
133 /// <para></para>
134 /// </summary>
135 public readonly IConverter<string, TLink> StringToUnicodeSequenceConverter;
136 /// <summary>
137 /// <para>
138 /// The unicode sequence to string converter.
139 /// </para>
140 /// <para></para>
141 /// </summary>
142 public readonly IConverter<TLink, string> UnicodeSequenceToStringConverter;
143 // For sequences
144 /// <summary>
145 /// <para>
146 /// The json array element criterion matcher.
147 /// </para>
148 /// <para></para>
149 /// </summary>
150 public readonly JsonArrayElementCriterionMatcher<TLink> JsonArrayElementCriterionMatcher;
151 /// <summary>
152 /// <para>
153 /// The default sequence right height provider.
154 /// </para>
155 /// <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>
275         ↳ listToSequenceConverter)
276     {
277         Links = links;
278         ListToSequenceConverter = listToSequenceConverter;
279         // Initializes constants
280         Any = Links.Constants.Any;
281         var markerIndex = One;
282         MeaningRoot = links.GetOrCreate(markerIndex, markerIndex);
283         var unicodeSymbolMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
284             ↳ markerIndex));
285         var unicodeSequenceMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
286             ↳ markerIndex));
287         DocumentMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
288             ↳ markerIndex));
289         ObjectMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
290         MemberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
291         ValueMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
292         StringMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
293         EmptyStringMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
294             ↳ markerIndex));
295         NumberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
296         NegativeNumberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
297             ↳ markerIndex));
298         ArrayMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
299         EmptyArrayMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
300             ↳ markerIndex));
301         TrueMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
302         FalseMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
303         NullMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
304         BalancedVariantConverter = new(links);
305         TargetMatcher<TLink> unicodeSymbolCriterionMatcher = new(Links, unicodeSymbolMarker);

```

```

299     TargetMatcher<TLink> unicodeSequenceCriterionMatcher = new(Links,
300         ↪ unicodeSequenceMarker);
301     CharToUnicodeSymbolConverter<TLink> charToUnicodeSymbolConverter =
302         new(Links, AddressToNumberConverter, unicodeSymbolMarker);
303     UnicodeSymbolToCharConverter<TLink> unicodeSymbolToCharConverter =
304         new(Links, NumberToAddressConverter, unicodeSymbolCriterionMatcher);
305     StringToUnicodeSequenceConverter = new CachingConverterDecorator<string, TLink>(
306         ↪ new StringToUnicodeSequenceConverter<TLink>(Links, charToUnicodeSymbolConverter,
307             ↪ BalancedVariantConverter, unicodeSequenceMarker));
308     RightSequenceWalker<TLink> sequenceWalker =
309         new(Links, new DefaultStack<TLink>(), unicodeSymbolCriterionMatcher.IsMatched);
310     UnicodeSequenceToStringConverter = new CachingConverterDecorator<TLink, string>(
311         ↪ new UnicodeSequenceToStringConverter<TLink>(Links,
312             ↪ unicodeSequenceCriterionMatcher, sequenceWalker,
313             ↪ unicodeSymbolToCharConverter));
314     BigIntegerToRawNumberSequenceConverter =
315         new(links, AddressToNumberConverter, ListToSequenceConverter,
316             ↪ NegativeNumberMarker);
317     RawNumberSequenceToBigIntegerConverter = new(links, NumberToAddressConverter,
318         ↪ NegativeNumberMarker);
319     DecimalToRationalConverter = new(links, BigIntegerToRawNumberSequenceConverter);
320     RationalToDecimalConverter = new(links, RawNumberSequenceToBigIntegerConverter);
321     JsonArrayElementCriterionMatcher = new(this);
322     DefaultSequenceRightHeightProvider = new(Links, JsonArrayElementCriterionMatcher);
323     DefaultSequenceAppender = new(Links, new DefaultStack<TLink>(),
324         ↪ DefaultSequenceRightHeightProvider);
325 }
326
327 /// <summary>
328 /// <para>
329 /// Creates the string using the specified content.
330 /// </para>
331 /// <para></para>
332 /// </summary>
333 /// <param name="content">
334 /// <para>The content.</para>
335 /// <para></para>
336 /// </param>
337 /// <returns>
338 /// <para>The link</para>
339 /// <para></para>
340 /// </returns>
341 public TLink CreateString(string content)
342 {
343     var @string = GetStringSequence(content);
344     return Links.GetOrCreate(StringMarker, @string);
345 }
346
347 /// <summary>
348 /// <para>
349 /// Creates the string value using the specified content.
350 /// </para>
351 /// <para></para>
352 /// </summary>
353 /// <param name="content">
354 /// <para>The content.</para>
355 /// <para></para>
356 /// </param>
357 /// <returns>
358 /// <para>The link</para>
359 /// <para></para>
360 /// </returns>
361 public TLink CreateStringValue(string content)
362 {
363     var @string = CreateString(content);
364     return CreateValue(@string);
365 }
366
367 /// <summary>
368 /// <para>
369 /// Creates the number using the specified number.
370 /// </para>
371 /// <para></para>
372 /// </summary>
373 /// <param name="number">
374 /// <para>The number.</para>
375 /// <para></para>
376 /// </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 :
499         ↪ BalancedVariantConverter.Convert(array);
500     return CreateArray(arraySequence);
501 }
502
503 /// <summary>
504 /// <para>
505 /// Creates the array using the specified sequence.
506 /// </para>
507 /// <para></para>
508 /// </summary>
509 /// <param name="sequence">
510 /// <para>The sequence.</para>
511 /// <para></para>
512 /// </param>
513 /// <returns>
514 /// <para>The link</para>
515 /// <para></para>
516 /// </returns>
517 public TLink CreateArray(TLink sequence) => Links.GetOrCreate(ArrayMarker, sequence);
518
519 /// <summary>
520 /// <para>
521 /// Creates the array value using the specified array.
522 /// </para>
523 /// <para></para>
524 /// </summary>
525 /// <param name="array">
526 /// <para>The array.</para>

```

```

526     /// <para></para>
527     /// </param>
528     /// <returns>
529     /// <para>The link</para>
530     /// <para></para>
531     /// </returns>
532     public TLink CreateArrayValue(ICollection<TLink> array)
533     {
534         var arrayLink = CreateArray(array);
535         return CreateValue(arrayLink);
536     }
537
538     /// <summary>
539     /// <para>
540     /// Creates the array value using the specified sequence.
541     /// </para>
542     /// <para></para>
543     /// </summary>
544     /// <param name="sequence">
545     /// <para>The sequence.</para>
546     /// <para></para>
547     /// </param>
548     /// <returns>
549     /// <para>The link</para>
550     /// <para></para>
551     /// </returns>
552     public TLink CreateArrayValue(TLink sequence)
553     {
554         var array = CreateArray(sequence);
555         return CreateValue(array);
556     }
557
558     /// <summary>
559     /// <para>
560     /// Creates the member using the specified name.
561     /// </para>
562     /// <para></para>
563     /// </summary>
564     /// <param name="name">
565     /// <para>The name.</para>
566     /// <para></para>
567     /// </param>
568     /// <returns>
569     /// <para>The link</para>
570     /// <para></para>
571     /// </returns>
572     public TLink CreateMember(string name)
573     {
574         var nameLink = CreateString(name);
575         return Links.GetOrCreate(MemberMarker, nameLink);
576     }
577
578     /// <summary>
579     /// <para>
580     /// Creates the value using the specified value.
581     /// </para>
582     /// <para></para>
583     /// </summary>
584     /// <param name="value">
585     /// <para>The value.</para>
586     /// <para></para>
587     /// </param>
588     /// <returns>
589     /// <para>The link</para>
590     /// <para></para>
591     /// </returns>
592     public TLink CreateValue(TLink value) => Links.GetOrCreate(ValueMarker, value);
593
594     /// <summary>
595     /// <para>
596     /// Attaches the object using the specified parent.
597     /// </para>
598     /// <para></para>
599     /// </summary>
600     /// <param name="parent">
601     /// <para>The parent.</para>
602     /// <para></para>
603     /// </param>

```



```

604     /// <returns>
605     /// <para>The link</para>
606     /// <para></para>
607     /// </returns>
608     public TLink AttachObject(TLink parent) => Attach(parent, CreateObjectValue());
609
610     /// <summary>
611     /// <para>
612     /// Attaches the string using the specified parent.
613     /// </para>
614     /// <para></para>
615     /// </summary>
616     /// <param name="parent">
617     /// <para>The parent.</para>
618     /// <para></para>
619     /// </param>
620     /// <param name="content">
621     /// <para>The content.</para>
622     /// <para></para>
623     /// </param>
624     /// <returns>
625     /// <para>The link</para>
626     /// <para></para>
627     /// </returns>
628     public TLink AttachString(TLink parent, string content)
629     {
630         var @string = CreateString(content);
631         var stringValue = CreateValue(@string);
632         return Attach(parent, stringValue);
633     }
634
635     /// <summary>
636     /// <para>
637     /// Attaches the number using the specified parent.
638     /// </para>
639     /// <para></para>
640     /// </summary>
641     /// <param name="parent">
642     /// <para>The parent.</para>
643     /// <para></para>
644     /// </param>
645     /// <param name="number">
646     /// <para>The number.</para>
647     /// <para></para>
648     /// </param>
649     /// <returns>
650     /// <para>The link</para>
651     /// <para></para>
652     /// </returns>
653     public TLink AttachNumber(TLink parent, decimal number)
654     {
655         var numberLink = CreateNumber(number);
656         var numberValue = CreateValue(numberLink);
657         return Attach(parent, numberValue);
658     }
659
660     /// <summary>
661     /// <para>
662     /// Attaches the boolean using the specified parent.
663     /// </para>
664     /// <para></para>
665     /// </summary>
666     /// <param name="parent">
667     /// <para>The parent.</para>
668     /// <para></para>
669     /// </param>
670     /// <param name="value">
671     /// <para>The value.</para>
672     /// <para></para>
673     /// </param>
674     /// <returns>
675     /// <para>The link</para>
676     /// <para></para>
677     /// </returns>
678     public TLink AttachBoolean(TLink parent, bool value)
679     {
680         var booleanValue = CreateBooleanValue(value);
681         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>

```

```

760     /// <para></para>
761     /// </param>
762     /// <param name="child">
763     /// <para>The child.</para>
764     /// <para></para>
765     /// </param>
766     /// <returns>
767     /// <para>The link</para>
768     /// <para></para>
769     /// </returns>
770     public TLink Attach(TLink parent, TLink child) => Links.GetOrCreate(parent, child);
771
772     /// <summary>
773     /// <para>
774     /// Appends the array value using the specified array value.
775     /// </para>
776     /// <para></para>
777     /// </summary>
778     /// <param name="arrayValue">
779     /// <para>The array value.</para>
780     /// <para></para>
781     /// </param>
782     /// <param name="appendant">
783     /// <para>The appendant.</para>
784     /// <para></para>
785     /// </param>
786     /// <returns>
787     /// <para>The new array value.</para>
788     /// <para></para>
789     /// </returns>
790     public TLink AppendArrayValue(TLink arrayValue, TLink appendant)
791     {
792         var array = GetArray(arrayValue);
793         var arraySequence = Links.GetTarget(array);
794         TLink newArrayValue;
795         if (EqualityComparer.Equals(arraySequence, EmptyArrayMarker))
796         {
797             newArrayValue = CreateArrayValue(appendant);
798         }
799         else
800         {
801             arraySequence = DefaultSequenceAppender.Append(arraySequence, appendant);
802             newArrayValue = CreateArrayValue(arraySequence);
803         }
804         return newArrayValue;
805     }
806
807     /// <summary>
808     /// <para>
809     /// Gets the document or default using the specified name.
810     /// </para>
811     /// <para></para>
812     /// </summary>
813     /// <param name="name">
814     /// <para>The name.</para>
815     /// <para></para>
816     /// </param>
817     /// <returns>
818     /// <para>The link</para>
819     /// <para></para>
820     /// </returns>
821     public TLink GetDocumentOrDefault(string name)
822     {
823         var stringSequence = GetStringSequence(name);
824         var @string = Links.SearchOrDefault(StringMarker, stringSequence);
825         if (EqualityComparer.Equals(@string, default))
826         {
827             return default;
828         }
829         return Links.SearchOrDefault(DocumentMarker, @string);
830     }
831
832     /// <summary>
833     /// <para>
834     /// Gets the string sequence using the specified content.
835     /// </para>
836     /// <para></para>
837     /// </summary>

```

```

838     /// <param name="content">
839     /// <para>The content.</para>
840     /// <para></para>
841     /// </param>
842     /// <returns>
843     /// <para>The link</para>
844     /// <para></para>
845     /// </returns>
846     private TLink GetStringSequence(string content) => content == "" ? EmptyStringMarker :
        ↳ StringToUnicodeSequenceConverter.Convert(content);

847
848     /// <summary>
849     /// <para>
850     /// Gets the string using the specified string value.
851     /// </para>
852     /// <para></para>
853     /// </summary>
854     /// <param name="stringValue">
855     /// <para>The string value.</para>
856     /// <para></para>
857     /// </param>
858     /// <exception cref="Exception">
859     /// <para>The passed link does not contain a string.</para>
860     /// <para></para>
861     /// </exception>
862     /// <returns>
863     /// <para>The string</para>
864     /// <para></para>
865     /// </returns>
866     public string GetString(TLink stringValue)
867     {
868         var current = stringValue;
869         TLink source;
870         for (int i = 0; i < 3; i++)
871         {
872             source = Links.GetSource(current);
873             if (EqualityComparer.Equals(source, StringMarker))
874             {
875                 var sequence = Links.GetTarget(current);
876                 var isEmpty = EqualityComparer.Equals(sequence, EmptyStringMarker);
877                 return isEmpty ? "" : UnicodeSequenceToStringConverter.Convert(sequence);
878             }
879             current = Links.GetTarget(current);
880         }
881         throw new Exception("The passed link does not contain a string.");
882     }

883
884     /// <summary>
885     /// <para>
886     /// Gets the number using the specified value link.
887     /// </para>
888     /// <para></para>
889     /// </summary>
890     /// <param name="valueLink">
891     /// <para>The value link.</para>
892     /// <para></para>
893     /// </param>
894     /// <exception cref="Exception">
895     /// <para>The passed link does not contain a number.</para>
896     /// <para></para>
897     /// </exception>
898     /// <returns>
899     /// <para>The decimal</para>
900     /// <para></para>
901     /// </returns>
902     public decimal GetNumber(TLink valueLink)
903     {
904         var current = valueLink;
905         TLink source;
906         TLink target;
907         for (int i = 0; i < 3; i++)
908         {
909             source = Links.GetSource(current);
910             target = Links.GetTarget(current);
911             if (EqualityComparer.Equals(source, NumberMarker))
912             {
913                 return RationalToDecimalConverter.Convert(target);
914             }

```

```

915         current = target;
916     }
917     throw new Exception("The passed link does not contain a number.");
918 }
919
920
921 /// <summary>
922 /// <para>
923 /// Gets the object using the specified object value link.
924 /// </para>
925 /// <para></para>
926 /// </summary>
927 /// <param name="objectValueLink">
928 /// <para>The object value link.</para>
929 /// <para></para>
930 /// </param>
931 /// <exception cref="Exception">
932 /// <para>The passed link does not contain an object.</para>
933 /// <para></para>
934 /// </exception>
935 /// <returns>
936 /// <para>The link</para>
937 /// <para></para>
938 /// </returns>
939 public TLink GetObject(TLink objectValueLink)
940 {
941     var current = objectValueLink;
942     TLink source;
943     for (int i = 0; i < 3; i++)
944     {
945         source = Links.GetSource(current);
946         if (EqualityComparer.Equals(source, ObjectMarker))
947         {
948             return current;
949         }
950         current = Links.GetTarget(current);
951     }
952     throw new Exception("The passed link does not contain an object.");
953 }
954
955 /// <summary>
956 /// <para>
957 /// Gets the array using the specified array value link.
958 /// </para>
959 /// <para></para>
960 /// </summary>
961 /// <param name="arrayValueLink">
962 /// <para>The array value link.</para>
963 /// <para></para>
964 /// </param>
965 /// <exception cref="Exception">
966 /// <para>The passed link does not contain an array.</para>
967 /// <para></para>
968 /// </exception>
969 /// <returns>
970 /// <para>The link</para>
971 /// <para></para>
972 /// </returns>
973 public TLink GetArray(TLink arrayValueLink)
974 {
975     var current = arrayValueLink;
976     TLink source;
977     for (int i = 0; i < 3; i++)
978     {
979         source = Links.GetSource(current);
980         if (EqualityComparer.Equals(source, ArrayMarker))
981         {
982             return current;
983         }
984         current = Links.GetTarget(current);
985     }
986     throw new Exception("The passed link does not contain an array.");
987 }
988
989 /// <summary>
990 /// <para>
991 /// Gets the array sequence using the specified array.
992 /// </para>

```

```

993     /// <para></para>
994     /// </summary>
995     /// <param name="array">
996     /// <para>The array.</para>
997     /// <para></para>
998     /// </param>
999     /// <returns>
1000    /// <para>The link</para>
1001    /// <para></para>
1002    /// </returns>
1003    public TLink GetArraySequence(TLink array) => Links.GetTarget(array);
1004
1005    /// <summary>
1006    /// <para>
1007    /// Gets the value link using the specified parent.
1008    /// </para>
1009    /// <para></para>
1010    /// </summary>
1011    /// <param name="parent">
1012    /// <para>The parent.</para>
1013    /// <para></para>
1014    /// </param>
1015    /// <exception cref="InvalidOperationException">
1016    /// <para>More than 1 value found.</para>
1017    /// <para></para>
1018    /// </exception>
1019    /// <exception cref="InvalidOperationException">
1020    /// <para>The list elements length is negative.</para>
1021    /// <para></para>
1022    /// </exception>
1023    /// <exception cref="InvalidOperationException">
1024    /// <para>The passed link is not a value.</para>
1025    /// <para></para>
1026    /// </exception>
1027    /// <returns>
1028    /// <para>The link</para>
1029    /// <para></para>
1030    /// </returns>
1031    public TLink GetValueLink(TLink parent)
1032    {
1033        var query = new Link<TLink>(index: Any, source: parent, target: Any);
1034        var resultLinks = Links.All(query);
1035        switch (resultLinks.Count)
1036        {
1037            case 0:
1038                return default;
1039            case 1:
1040                var resultLinkTarget = Links.GetTarget(resultLinks[0]);
1041                if (EqualityComparer.Equals(Links.GetSource(resultLinkTarget), ValueMarker))
1042                {
1043                    return resultLinkTarget;
1044                }
1045                else
1046                {
1047                    throw new InvalidOperationException("The passed link is not a value.");
1048                }
1049            case > 1:
1050                throw new InvalidOperationException("More than 1 value found.");
1051            default:
1052                throw new InvalidOperationException("The list elements length is negative.");
1053        }
1054    }
1055
1056    /// <summary>
1057    /// <para>
1058    /// Gets the value marker using the specified value.
1059    /// </para>
1060    /// <para></para>
1061    /// </summary>
1062    /// <param name="value">
1063    /// <para>The value.</para>
1064    /// <para></para>
1065    /// </param>
1066    /// <returns>
1067    /// <para>The target source.</para>
1068    /// <para></para>
1069    /// </returns>
1070    public TLink GetValueMarker(TLink value)

```

```

1071     {
1072         var target = Links.GetTarget(value);
1073         var targetSource = Links.GetSource(target);
1074         if (EqualityComparer.Equals(MeaningRoot, targetSource))
1075         {
1076             return target;
1077         }
1078         return targetSource;
1079     }
1080
1081     /// <summary>
1082     /// <para>
1083     /// Gets the members links using the specified object.
1084     /// </para>
1085     /// <para></para>
1086     /// </summary>
1087     /// <param name="@object">
1088     /// <para>The object.</para>
1089     /// <para></para>
1090     /// </param>
1091     /// <returns>
1092     /// <para>The members.</para>
1093     /// <para></para>
1094     /// </returns>
1095     public List<TLink> GetMembersLinks(TLink @object)
1096     {
1097         Link<TLink> query = new(index: Any, source: @object, target: Any);
1098         List<TLink> members = new();
1099         Links.Each(objectMemberLink =>
1100         {
1101             var memberLink = Links.GetTarget(objectMemberLink);
1102             var memberMarker = Links.GetSource(memberLink);
1103             if (EqualityComparer.Equals(memberMarker, MemberMarker))
1104             {
1105                 members.Add(Links.GetIndex(objectMemberLink));
1106             }
1107             return Links.Constants.Continue;
1108         }, query);
1109         return members;
1110     }
1111 }
1112 }

```

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

```

1 using System.Collections.Generic;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.Data.Doublets.Json
6 {
7     /// <summary>
8     /// <para>
9     /// Defines the json storage.
10    /// </para>
11    /// <para></para>
12    /// </summary>
13    public interface IJsonStorage<TLink>
14    {
15        /// <summary>
16        /// <para>
17        /// Gets the links value.
18        /// </para>
19        /// <para></para>
20        /// </summary>
21        public ILinks<TLink> Links { get; }
22        /// <summary>
23        /// <para>
24        /// Gets the document marker value.
25        /// </para>
26        /// <para></para>
27        /// </summary>
28        public TLink DocumentMarker { get; }
29        /// <summary>
30        /// <para>
31        /// Gets the object marker value.
32        /// </para>
33        /// <para></para>
34        /// </summary>
35        public TLink ObjectMarker { get; }

```

```

36     /// <summary>
37     /// <para>
38     /// Gets the string marker value.
39     /// </para>
40     /// <para></para>
41     /// </summary>
42     public TLink StringMarker { get; }
43     /// <summary>
44     /// <para>
45     /// Gets the empty string marker value.
46     /// </para>
47     /// <para></para>
48     /// </summary>
49     public TLink EmptyStringMarker { get; }
50     /// <summary>
51     /// <para>
52     /// Gets the member marker value.
53     /// </para>
54     /// <para></para>
55     /// </summary>
56     public TLink MemberMarker { get; }
57     /// <summary>
58     /// <para>
59     /// Gets the value marker value.
60     /// </para>
61     /// <para></para>
62     /// </summary>
63     public TLink ValueMarker { get; }
64     /// <summary>
65     /// <para>
66     /// Gets the number marker value.
67     /// </para>
68     /// <para></para>
69     /// </summary>
70     public TLink NumberMarker { get; }
71     /// <summary>
72     /// <para>
73     /// Gets the array marker value.
74     /// </para>
75     /// <para></para>
76     /// </summary>
77     public TLink ArrayMarker { get; }
78     /// <summary>
79     /// <para>
80     /// Gets the empty array marker value.
81     /// </para>
82     /// <para></para>
83     /// </summary>
84     public TLink EmptyArrayMarker { get; }
85     /// <summary>
86     /// <para>
87     /// Gets the true marker value.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     public TLink TrueMarker { get; }
92     /// <summary>
93     /// <para>
94     /// Gets the false marker value.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     public TLink FalseMarker { get; }
99     /// <summary>
100    /// <para>
101    /// Gets the null marker value.
102    /// </para>
103    /// <para></para>
104    /// </summary>
105    public TLink NullMarker { get; }
106    /// <summary>
107    /// <para>
108    /// Creates the string using the specified content.
109    /// </para>
110    /// <para></para>
111    /// </summary>
112    /// <param name="content">
113    /// <para>The content.</para>

```



```

114    /// <para></para>
115    /// </param>
116    /// <returns>
117    /// <para>The link</para>
118    /// <para></para>
119    /// </returns>
120    TLink CreateString(string content);
121    /// <summary>
122    /// <para>
123    /// Creates the string value using the specified content.
124    /// </para>
125    /// <para></para>
126    /// </summary>
127    /// <param name="content">
128    /// <para>The content.</para>
129    /// <para></para>
130    /// </param>
131    /// <returns>
132    /// <para>The link</para>
133    /// <para></para>
134    /// </returns>
135    TLink CreateStringValue(string content);
136    /// <summary>
137    /// <para>
138    /// Creates the number using the specified number.
139    /// </para>
140    /// <para></para>
141    /// </summary>
142    /// <param name="number">
143    /// <para>The number.</para>
144    /// <para></para>
145    /// </param>
146    /// <returns>
147    /// <para>The link</para>
148    /// <para></para>
149    /// </returns>
150    TLink CreateNumber(decimal number);
151    /// <summary>
152    /// <para>
153    /// Creates the number value using the specified number.
154    /// </para>
155    /// <para></para>
156    /// </summary>
157    /// <param name="number">
158    /// <para>The number.</para>
159    /// <para></para>
160    /// </param>
161    /// <returns>
162    /// <para>The link</para>
163    /// <para></para>
164    /// </returns>
165    TLink CreateNumberValue(decimal number);
166    /// <summary>
167    /// <para>
168    /// Creates the boolean value using the specified value.
169    /// </para>
170    /// <para></para>
171    /// </summary>
172    /// <param name="value">
173    /// <para>The value.</para>
174    /// <para></para>
175    /// </param>
176    /// <returns>
177    /// <para>The link</para>
178    /// <para></para>
179    /// </returns>
180    TLink CreateBooleanValue(bool value);
181    /// <summary>
182    /// <para>
183    /// Creates the null value.
184    /// </para>
185    /// <para></para>
186    /// </summary>
187    /// <returns>
188    /// <para>The link</para>
189    /// <para></para>
190    /// </returns>
191    TLink CreateNullValue();

```

```

192     /// <summary>
193     /// <para>
194     /// Creates the document using the specified name.
195     /// </para>
196     /// <para></para>
197     /// </summary>
198     /// <param name="name">
199     /// <para>The name.</para>
200     /// <para></para>
201     /// </param>
202     /// <returns>
203     /// <para>The link</para>
204     /// <para></para>
205     /// </returns>
206 TLink CreateDocument(string name);
207     /// <summary>
208     /// <para>
209     /// Gets the document or default using the specified name.
210     /// </para>
211     /// <para></para>
212     /// </summary>
213     /// <param name="name">
214     /// <para>The name.</para>
215     /// <para></para>
216     /// </param>
217     /// <returns>
218     /// <para>The link</para>
219     /// <para></para>
220     /// </returns>
221 TLink GetDocumentOrDefault(string name);
222     /// <summary>
223     /// <para>
224     /// Creates the object.
225     /// </para>
226     /// <para></para>
227     /// </summary>
228     /// <returns>
229     /// <para>The link</para>
230     /// <para></para>
231     /// </returns>
232 TLink CreateObject();
233     /// <summary>
234     /// <para>
235     /// Creates the object value.
236     /// </para>
237     /// <para></para>
238     /// </summary>
239     /// <returns>
240     /// <para>The link</para>
241     /// <para></para>
242     /// </returns>
243 TLink CreateObjectValue();
244     /// <summary>
245     /// <para>
246     /// Creates the array using the specified array.
247     /// </para>
248     /// <para></para>
249     /// </summary>
250     /// <param name="array">
251     /// <para>The array.</para>
252     /// <para></para>
253     /// </param>
254     /// <returns>
255     /// <para>The link</para>
256     /// <para></para>
257     /// </returns>
258 TLink CreateArray(IList<TLink> array);
259     /// <summary>
260     /// <para>
261     /// Creates the array value using the specified array.
262     /// </para>
263     /// <para></para>
264     /// </summary>
265     /// <param name="array">
266     /// <para>The array.</para>
267     /// <para></para>
268     /// </param>
269     /// <returns>

```

```

270    /// <para>The link</para>
271    /// <para></para>
272    /// </returns>
273    TLink CreateArrayValue(IList<TLink> array) => CreateValue(CreateArray(array));
274    /// <summary>
275    /// <para>
276    /// Creates the array value using the specified array.
277    /// </para>
278    /// <para></para>
279    /// </summary>
280    /// <param name="array">
281    /// <para>The array.</para>
282    /// <para></para>
283    /// </param>
284    /// <returns>
285    /// <para>The link</para>
286    /// <para></para>
287    /// </returns>
288    TLink CreateArrayValue(TLink array) => CreateValue(array);
289    /// <summary>
290    /// <para>
291    /// Creates the member using the specified name.
292    /// </para>
293    /// <para></para>
294    /// </summary>
295    /// <param name="name">
296    /// <para>The name.</para>
297    /// <para></para>
298    /// </param>
299    /// <returns>
300    /// <para>The link</para>
301    /// <para></para>
302    /// </returns>
303    TLink CreateMember(string name);
304    /// <summary>
305    /// <para>
306    /// Creates the value using the specified value.
307    /// </para>
308    /// <para></para>
309    /// </summary>
310    /// <param name="value">
311    /// <para>The value.</para>
312    /// <para></para>
313    /// </param>
314    /// <returns>
315    /// <para>The link</para>
316    /// <para></para>
317    /// </returns>
318    TLink CreateValue(TLink value);
319    /// <summary>
320    /// <para>
321    /// Attaches the source.
322    /// </para>
323    /// <para></para>
324    /// </summary>
325    /// <param name="source">
326    /// <para>The source.</para>
327    /// <para></para>
328    /// </param>
329    /// <param name="target">
330    /// <para>The target.</para>
331    /// <para></para>
332    /// </param>
333    /// <returns>
334    /// <para>The link</para>
335    /// <para></para>
336    /// </returns>
337    TLink Attach(TLink source, TLink target);
338    /// <summary>
339    /// <para>
340    /// Attaches the object using the specified parent.
341    /// </para>
342    /// <para></para>
343    /// </summary>
344    /// <param name="parent">
345    /// <para>The parent.</para>
346    /// <para></para>
347    /// </param>

```

```

348     /// <returns>
349     /// <para>The link</para>
350     /// <para></para>
351     /// </returns>
352     TLink AttachObject(TLink parent);
353     /// <summary>
354     /// <para>
355     /// Attaches the string using the specified parent.
356     /// </para>
357     /// <para></para>
358     /// </summary>
359     /// <param name="parent">
360     /// <para>The parent.</para>
361     /// <para></para>
362     /// </param>
363     /// <param name="content">
364     /// <para>The content.</para>
365     /// <para></para>
366     /// </param>
367     /// <returns>
368     /// <para>The link</para>
369     /// <para></para>
370     /// </returns>
371     TLink AttachString(TLink parent, string content);
372     /// <summary>
373     /// <para>
374     /// Attaches the number using the specified parent.
375     /// </para>
376     /// <para></para>
377     /// </summary>
378     /// <param name="parent">
379     /// <para>The parent.</para>
380     /// <para></para>
381     /// </param>
382     /// <param name="number">
383     /// <para>The number.</para>
384     /// <para></para>
385     /// </param>
386     /// <returns>
387     /// <para>The link</para>
388     /// <para></para>
389     /// </returns>
390     TLink AttachNumber(TLink parent, decimal number);
391     /// <summary>
392     /// <para>
393     /// Attaches the boolean using the specified parent.
394     /// </para>
395     /// <para></para>
396     /// </summary>
397     /// <param name="parent">
398     /// <para>The parent.</para>
399     /// <para></para>
400     /// </param>
401     /// <param name="value">
402     /// <para>The value.</para>
403     /// <para></para>
404     /// </param>
405     /// <returns>
406     /// <para>The link</para>
407     /// <para></para>
408     /// </returns>
409     TLink AttachBoolean(TLink parent, bool value);
410     /// <summary>
411     /// <para>
412     /// Attaches the null using the specified parent.
413     /// </para>
414     /// <para></para>
415     /// </summary>
416     /// <param name="parent">
417     /// <para>The parent.</para>
418     /// <para></para>
419     /// </param>
420     /// <returns>
421     /// <para>The link</para>
422     /// <para></para>
423     /// </returns>
424     TLink AttachNull(TLink parent);
425     /// <summary>

```

```

426     /// <para>
427     /// Attaches the array using the specified parent.
428     /// </para>
429     /// <para></para>
430     /// </summary>
431     /// <param name="parent">
432     /// <para>The parent.</para>
433     /// <para></para>
434     /// </param>
435     /// <param name="array">
436     /// <para>The array.</para>
437     /// <para></para>
438     /// </param>
439     /// <returns>
440     /// <para>The link</para>
441     /// <para></para>
442     /// </returns>
443     TLink AttachArray(TLink parent, IList<TLink> array);
444     /// <summary>
445     /// <para>
446     /// Attaches the member to object using the specified object.
447     /// </para>
448     /// <para></para>
449     /// </summary>
450     /// <param name="@object">
451     /// <para>The object.</para>
452     /// <para></para>
453     /// </param>
454     /// <param name="keyName">
455     /// <para>The key name.</para>
456     /// <para></para>
457     /// </param>
458     /// <returns>
459     /// <para>The link</para>
460     /// <para></para>
461     /// </returns>
462     TLink AttachMemberToObject(TLink @object, string keyName);
463     /// <summary>
464     /// <para>
465     /// Appends the array value using the specified array value.
466     /// </para>
467     /// <para></para>
468     /// </summary>
469     /// <param name="arrayValue">
470     /// <para>The array value.</para>
471     /// <para></para>
472     /// </param>
473     /// <param name="appendant">
474     /// <para>The appendant.</para>
475     /// <para></para>
476     /// </param>
477     /// <returns>
478     /// <para>The link</para>
479     /// <para></para>
480     /// </returns>
481     TLink AppendArrayValue(TLink arrayValue, TLink appendant);
482     /// <summary>
483     /// <para>
484     /// Gets the string using the specified string value.
485     /// </para>
486     /// <para></para>
487     /// </summary>
488     /// <param name="stringValue">
489     /// <para>The string value.</para>
490     /// <para></para>
491     /// </param>
492     /// <returns>
493     /// <para>The string</para>
494     /// <para></para>
495     /// </returns>
496     string GetString(TLink stringValue);
497     /// <summary>
498     /// <para>
499     /// Gets the number using the specified value.
500     /// </para>
501     /// <para></para>
502     /// </summary>
503     /// <param name="value">

```

```

504    /// <para>The value.</para>
505    /// <para></para>
506    /// </param>
507    /// <returns>
508    /// <para>The decimal</para>
509    /// <para></para>
510    /// </returns>
511    decimal GetNumber(TLink value);
512    /// <summary>
513    /// <para>
514    /// Gets the object using the specified object value.
515    /// </para>
516    /// <para></para>
517    /// </summary>
518    /// <param name="objectValue">
519    /// <para>The object value.</para>
520    /// <para></para>
521    /// </param>
522    /// <returns>
523    /// <para>The link</para>
524    /// <para></para>
525    /// </returns>
526    TLink GetObject(TLink objectValue);
527    /// <summary>
528    /// <para>
529    /// Gets the array using the specified array value link.
530    /// </para>
531    /// <para></para>
532    /// </summary>
533    /// <param name="arrayValueLink">
534    /// <para>The array value link.</para>
535    /// <para></para>
536    /// </param>
537    /// <returns>
538    /// <para>The link</para>
539    /// <para></para>
540    /// </returns>
541    TLink GetArray(TLink arrayValueLink);
542    /// <summary>
543    /// <para>
544    /// Gets the array sequence using the specified array.
545    /// </para>
546    /// <para></para>
547    /// </summary>
548    /// <param name="array">
549    /// <para>The array.</para>
550    /// <para></para>
551    /// </param>
552    /// <returns>
553    /// <para>The link</para>
554    /// <para></para>
555    /// </returns>
556    TLink GetArraySequence(TLink array);
557    /// <summary>
558    /// <para>
559    /// Gets the value link using the specified parent.
560    /// </para>
561    /// <para></para>
562    /// </summary>
563    /// <param name="parent">
564    /// <para>The parent.</para>
565    /// <para></para>
566    /// </param>
567    /// <returns>
568    /// <para>The link</para>
569    /// <para></para>
570    /// </returns>
571    TLink GetValueLink(TLink parent);
572    /// <summary>
573    /// <para>
574    /// Gets the value marker using the specified link.
575    /// </para>
576    /// <para></para>
577    /// </summary>
578    /// <param name="link">
579    /// <para>The link.</para>
580    /// <para></para>
581    /// </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 =
47             ↪ storage;
48         /// <summary>
49         /// <para>
50         /// Determines whether this instance is matched.
51         /// </para>
52         /// <para></para>
53         /// </summary>
54         /// <param name="link">
55         /// <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 }

```



```

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) =>
82         ↪ utf8JsonWriter.WriteStringValue(Storage.GetString(valueLink));
83
84     /// <summary>
85     /// <para>
86     /// Writes the string using the specified utf 8 json writer.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="utf8JsonWriter">
91     /// <para>The utf json writer.</para>
92     /// <para></para>
93     /// </param>
94     /// <param name="parent">
95     /// <para>The parent.</para>
96     /// <para></para>
97     /// </param>
98     /// <param name="valueLink">
99     /// <para>The value link.</para>
100    /// <para></para>
101    /// </param>
102    private void WriteString(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
103        ↪ valueLink) => utf8JsonWriter.WriteString(parent, Storage.GetString(valueLink));
104
105    /// <summary>
106    /// <para>
107    /// Writes the number value using the specified utf 8 json writer.
108    /// </para>
109    /// <para></para>
110    /// </summary>
111    /// <param name="utf8JsonWriter">
112    /// <para>The utf json writer.</para>
113    /// <para></para>
114    /// </param>
115    /// <param name="valueLink">
116    /// <para>The value link.</para>
117    /// <para></para>
118    /// </param>
119    private void WriteNumberValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
120        ↪ utf8JsonWriter.WriteNumberValue(Storage.GetNumber(valueLink));
121
122    /// <summary>
123    /// <para>
124    /// Writes the number using the specified utf 8 json writer.
125    /// </para>
126    /// <para></para>
127    /// </summary>
128    /// <param name="utf8JsonWriter">
129    /// <para>The utf json writer.</para>
130    /// <para></para>
131    /// </param>
132    /// <param name="parent">
133    /// <para>The parent.</para>
134    /// <para></para>
135    /// </param>
136    /// <param name="valueLink">
137    /// <para>The value link.</para>
138    /// <para></para>
139    /// </param>
140    private void WriteNumber(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
141        ↪ valueLink) => utf8JsonWriter.WriteNumber(parent, Storage.GetNumber(valueLink));
142
143    /// <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,
    → CancellationTokens 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
    ↪ Cancellation token 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);

```

```

290     }
291     else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
292     {
293         utf8JsonWriter.WriteBooleanValue(true);
294     }
295     else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
296     {
297         utf8JsonWriter.WriteBooleanValue(false);
298     }
299     else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
300     {
301         utf8JsonWriter.WriteNullValue();
302     }
303 }
304
305 /// <summary>
306 /// <para>
307 /// Exports the document.
308 /// </para>
309 /// <para></para>
310 /// </summary>
311 /// <param name="document">
312 /// <para>The document.</para>
313 /// <para></para>
314 /// </param>
315 /// <param name="utf8JsonWriter">
316 /// <para>The utf json writer.</para>
317 /// <para></para>
318 /// </param>
319 /// <param name="cancellationToken">
320 /// <para>The cancellation token.</para>
321 /// <para></para>
322 /// </param>
323 /// <exception cref="Exception">
324 /// <para>No document with this name exists</para>
325 /// <para></para>
326 /// </exception>
327 public void Export(TLink document, ref Utf8JsonWriter utf8JsonWriter, in
    ↳ Cancellation token cancellationToken)
328 {
329     if (EqualityComparer.Equals(document, default))
330     {
331         throw new Exception("No document with this name exists");
332     }
333     var valueLink = Storage.GetValueLink(document);
334     Write(ref utf8JsonWriter, valueLink, in cancellationToken);
335     utf8JsonWriter.Flush();
336 }
337
338 /// <summary>
339 /// <para>
340 /// Exports the document name.
341 /// </para>
342 /// <para></para>
343 /// </summary>
344 /// <param name="documentName">
345 /// <para>The document name.</para>
346 /// <para></para>
347 /// </param>
348 /// <param name="utf8JsonWriter">
349 /// <para>The utf json writer.</para>
350 /// <para></para>
351 /// </param>
352 /// <param name="cancellationToken">
353 /// <para>The cancellation token.</para>
354 /// <para></para>
355 /// </param>
356 public void Export(string documentName, Utf8JsonWriter utf8JsonWriter, Cancellation token
    ↳ cancellationToken) => Export(Storage.GetDocumentOrDefault(documentName), ref
    ↳ utf8JsonWriter, in cancellationToken);
357 }
358 }

```

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

```

1 using System;
2 using System.IO;
3 using System.Text.Encodings.Web;
4 using Platform.Data.Doublets.Memory.United.Generic;

```

```

5 using Platform.IO;
6 using System.Text.Json;
7 using Platform.Data.Doublets.Memory;
8 using Platform.Data.Doublets.Sequences.Converters;
9 using Platform.Memory;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.Json
14 {
15     /// <summary>
16     /// <para>
17     /// Represents the json exporter cli.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     public class JsonExporterCli<TLink>
22         where TLink : struct
23     {
24         /// <summary>
25         /// <para>
26         /// Runs the args.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         /// <param name="args">
31         /// <para>The args.</para>
32         /// <para></para>
33         /// </param>
34         public void Run(params string[] args)
35         {
36             var argumentIndex = 0;
37             var linksFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Links file
38                 ↳ path", args);
39             var jsonFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "JSON file
40                 ↳ path", args);
41             var defaultDocumentName = Path.GetFileNameWithoutExtension(jsonFilePath);
42             var documentName = ConsoleHelpers.GetOrReadArgument(argumentIndex, $"Document name
43                 ↳ (default: {defaultDocumentName})", args);
44             if (string.IsNullOrEmpty(documentName))
45             {
46                 documentName = defaultDocumentName;
47             }
48             if (!File.Exists(linksFilePath))
49             {
50                 Console.WriteLine($"${linksFilePath} file does not exist.");
51             }
52             using FileStream jsonFileStream = new(jsonFilePath, FileMode.Append);
53             JsonSerializerOptions utf8JsonWriterOptions = new()
54             {
55                 Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping,
56                 Indented = true
57             };
58             Utf8JsonWriter utf8JsonWriter = new(jsonFileStream, utf8JsonWriterOptions);
59             var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
60                 ↳ true);
61             using UnitedMemoryLinks<TLink> memoryAdapter = new (new
62                 ↳ FileMappedResizableDirectMemory(linksFilePath),
63                 ↳ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
64                 ↳ IndexTreeType.Default);
65             var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
66             BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
67             var storage = new DefaultJsonStorage<TLink>(links, balancedVariantConverter);
68             var exporter = new JsonExporter<TLink>(storage);
69             var document = storage.GetDocumentOrDefault(documentName);
70             if (storage.EqualityComparer.Equals(document, default))
71             {
72                 Console.WriteLine("No document with this name.");
73             }
74             using ConsoleCancellation cancellation = new ();
75             var cancellationToken = cancellation.Token;
76             Console.WriteLine("Press CTRL+C to stop.");
77             try
78             {
79                 exporter.Export(document, ref utf8JsonWriter, in cancellationToken);
80             }
81             catch (Exception exception)
82             {
83             }
84         }
85     }
86 }

```

```

76         Console.WriteLine(exception);
77         return;
78     }
79     finally
80     {
81         utf8JsonWriter.Dispose();
82     }
83     Console.WriteLine("Export completed successfully.");
84 }
85 }
86 }

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text.Json;
4  using System.Threading;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.Data.Doublets.Json
9  {
10     /// <summary>
11     /// <para>
12     /// Represents the json importer.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public class JsonImporter<TLink>
17     {
18         /// <summary>
19         /// <para>
20         /// The storage.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         public readonly IJsonStorage<TLink> Storage;
25         /// <summary>
26         /// <para>
27         /// The default.
28         /// </para>
29         /// <para></para>
30         /// </summary>
31         public readonly EqualityComparer<TLink> EqualityComparer =
32             ↪ EqualityComparer<TLink>.Default;
33         /// <summary>
34         /// <para>
35         /// The parents.
36         /// </para>
37         /// <para></para>
38         /// </summary>
39         public readonly Stack<TLink> Parents = new ();
40         /// <summary>
41         /// <para>
42         /// Initializes a new <see cref="JsonImporter"/> instance.
43         /// </para>
44         /// <para></para>
45         /// </summary>
46         /// <param name="storage">
47         /// <para>A storage.</para>
48         /// </param>
49         public JsonImporter(IJsonStorage<TLink> storage) => Storage = storage;
50
51         /// <summary>
52         /// <para>
53         /// Pops the if parent is member.
54         /// </para>
55         /// <para></para>
56         /// </summary>
57         private void PopIfParentIsMember()
58         {
59             var parent = Parents.Peek();
60             var parentMarker = Storage.GetValueMarker(parent);
61             if (EqualityComparer.Equals(parentMarker, Storage.MemberMarker))
62             {
63                 Parents.Pop();
64             }
65         }
66     }
67 }

```

```

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
94     ↪ Cancellation token cancellationToken)
95     {
96         Parents.Clear();
97         if (!EqualityComparer.Equals(Storage.GetDocumentOrDefault(documentName), default))
98         {
99             throw new Exception("The document with the specified name already exists.");
100         }
101         var document = Storage.CreateDocument(documentName);
102         Parents.Push(document);
103         TLink parent;
104         TLink parentMarker;
105         JsonTokenType tokenType;
106         TLink value;
107         TLink newParentArray;
108         while (utf8JsonReader.Read())
109         {
110             cancellationToken.ThrowIfCancellationRequested();
111             parent = Parents.Peek();
112             parentMarker = Storage.GetValueMarker(parent);
113             tokenType = utf8JsonReader.TokenType;
114             if (utf8JsonReader.TokenType == JsonTokenType.PropertyName)
115             {
116                 var @object = Storage.GetObject(parent);
117                 var property = utf8JsonReader.GetString();
118                 Parents.Push(Storage.AttachMemberToObject(@object, property));
119             }
120             switch (tokenType)
121             {
122                 case JsonTokenType.StartObject:
123                 {
124                     value = Storage.CreateObjectValue();
125                     if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
126                     {
127                         Parents.Pop();
128                         newParentArray = Storage.AppendArrayValue(parent, value);
129                         Parents.Push(newParentArray);
130                         Parents.Push(value);
131                     }
132                     else
133                     {
134                         var @object = Storage.Attach(parent, value);
135                         Parents.Push(@object);
136                     }
137                     break;
138                 }
139                 case JsonTokenType.EndObject:
140                 {
141                     Parents.Pop();
142                     break;
143                 }
144                 case JsonTokenType.StartArray:
145                 {
146                     value = Storage.CreateArrayValue(Array.Empty<TLink>());
147                 }
148             }
149         }
150     }

```

```

143     Parents.Push(value);
144     break;
145 case JsonTokenType.EndArray:
146 {
147     var arrayValue = Parents.Pop();
148     parent = Parents.Peek();
149     parentMarker = Storage.GetValueMarker(parent);
150     if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
151     {
152         Parents.Pop();
153         newParentArray = Storage.AppendArrayValue(parent, arrayValue);
154         Parents.Push(newParentArray);
155     }
156     Storage.Attach(parent, arrayValue);
157     break;
158 }
159 case JsonTokenType.String:
160 {
161     var @string = utf8JsonReader.GetString();
162     value = Storage.CreateStringValue(@string);
163     if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
164     {
165         Parents.Pop();
166         newParentArray = Storage.AppendArrayValue(parent, value);
167         Parents.Push(newParentArray);
168     }
169     else
170     {
171         Storage.Attach(parent, value);
172     }
173     break;
174 }
175 case JsonTokenType.Number:
176 {
177     value = Storage.CreateNumberValue(utf8JsonReader.GetDecimal());
178     if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
179     {
180         Parents.Pop();
181         newParentArray = Storage.AppendArrayValue(parent, value);
182         Parents.Push(newParentArray);
183     }
184     else
185     {
186         Storage.Attach(parent, value);
187     }
188     break;
189 }
190 case JsonTokenType.True:
191 {
192     value = Storage.CreateBooleanValue(true);
193     if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
194     {
195         Parents.Pop();
196         newParentArray = Storage.AppendArrayValue(parent, value);
197         Parents.Push(newParentArray);
198     }
199     else
200     {
201         Storage.Attach(parent, value);
202     }
203     break;
204 }
205 case JsonTokenType.False:
206 {
207     value = Storage.CreateBooleanValue(false);
208     if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
209     {
210         Parents.Pop();
211         newParentArray = Storage.AppendArrayValue(parent, value);
212         Parents.Push(newParentArray);
213     }
214     else
215     {
216         Storage.Attach(parent, value);
217     }
218     break;
219 }
220 case JsonTokenType.Null:
221 {

```



```

222         value = Storage.CreateNullValue();
223         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
224         {
225             Parents.Pop();
226             newParentArray = Storage.AppendArrayValue(parent, value);
227             Parents.Push(newParentArray);
228         }
229         else
230         {
231             Storage.Attach(parent, value);
232         }
233         break;
234     }
235 }
236 if (tokenType != JsonTokenType.PropertyName && tokenType !=
↪   JsonTokenType.StartObject && tokenType != JsonTokenType.StartArray)
237 {
238     PopIfParentIsMember();
239 }
240 }
241 return document;
242 }
243 }
244 }

```

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

```

1  using System;
2  using System.IO;
3  using System.Text;
4  using Platform.Data.Doublets.Memory.United.Generic;
5  using Platform.IO;
6  using System.Text.Json;
7  using Platform.Data.Doublets.Memory;
8  using Platform.Data.Doublets.Sequences.Converters;
9  using Platform.Memory;
10
11 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
12
13 namespace Platform.Data.Doublets.Json
14 {
15     /// <summary>
16     /// <para>
17     /// Represents the json importer cli.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     public class JsonImporterCli<TLink>
22         where TLink : struct
23     {
24         /// <summary>
25         /// <para>
26         /// Runs the args.
27         /// </para>
28         /// <para></para>
29         /// </summary>
30         /// <param name="args">
31         /// <para>The args.</para>
32         /// <para></para>
33         /// </param>
34         public void Run(params string[] args)
35         {
36             var argumentIndex = 0;
37             var jsonFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "JSON file
↪   path", args);
38             var linksFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Links file
↪   path", args);
39             var defaultDocumentName = Path.GetFileNameWithoutExtension(jsonFilePath);
40             var documentName = ConsoleHelpers.GetOrReadArgument(argumentIndex, $"Document name
↪   (default: {defaultDocumentName})", args);
41             if (string.IsNullOrEmpty(documentName))
42             {
43                 documentName = defaultDocumentName;
44             }
45             if (!File.Exists(jsonFilePath))
46             {
47                 Console.WriteLine($"'{jsonFilePath}' file does not exist.");
48             }
49             var json = File.ReadAllText(jsonFilePath);
50             var encodedJson = Encoding.UTF8.GetBytes(json);

```

```

51     ReadOnlySpan<byte> readOnlySpanEncodedJson = new(encodedJson);
52     Utf8JsonReader utf8JsonReader = new(readOnlySpanEncodedJson);
53     LinksConstants<TLink> linksConstants = new(enableExternalReferencesSupport: true);
54     FileMappedResizableDirectMemory fileMappedResizableDirectMemory = new(linksFilePath);
55     var unitedMemoryLinks = UnitedMemoryLinks<TLink>.DefaultLinksSizeStep;
56     const IndexTreeType indexTreeType = IndexTreeType.Default;
57     using UnitedMemoryLinks<TLink> memoryAdapter = new(fileMappedResizableDirectMemory,
58         ↪ unitedMemoryLinks, linksConstants, indexTreeType);
59     var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
60     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
61     DefaultJsonStorage<TLink> storage = new(links, balancedVariantConverter);
62     JsonImporter<TLink> importer = new(storage);
63     using ConsoleCancellation cancellation = new();
64     var cancellationToken = cancellation.Token;
65     Console.WriteLine("Press CTRL+C to stop.");
66     try
67     {
68         importer.Import(documentName, ref utf8JsonReader, in cancellationToken);
69     }
70     catch (Exception exception)
71     {
72         Console.WriteLine(exception);
73         return;
74     }
75     Console.WriteLine("Import completed successfully.");
76 }
77 }

```

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

```

1  using System.Text;
2  using System.Text.Json;
3  using System.Threading;
4  using System.IO;
5  using Xunit;
6  using TLink = System.UInt64;
7  using Platform.Data.Doublets.Memory.United.Generic;
8  using Platform.Memory;
9  using Platform.Data.Doublets.Memory;
10 using System.Text.RegularExpressions;
11 using Platform.Data.Doublets.Sequences.Converters;
12
13 namespace Platform.Data.Doublets.Json.Tests
14 {
15     /// <summary>
16     /// <para>
17     /// Represents the json import and export tests.
18     /// </para>
19     /// <para></para>
20     /// </summary>
21     public class JsonImportAndExportTests
22     {
23         /// <summary>
24         /// <para>
25         /// The balanced variant converter.
26         /// </para>
27         /// <para></para>
28         /// </summary>
29         public static BalancedVariantConverter<TLink> BalancedVariantConverter;
30
31         /// <summary>
32         /// <para>
33         /// Creates the links.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         /// <returns>
38         /// <para>A links of t link</para>
39         /// <para></para>
40         /// </returns>
41         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new IO.TemporaryFile());
42
43         /// <summary>
44         /// <para>
45         /// Creates the links using the specified data db filename.
46         /// </para>
47         /// <para></para>
48         /// </summary>
49         /// <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:
64            ↪ true);
65        return new UnitedMemoryLinks<TLink>(new
66            ↪ FileMappedResizableDirectMemory(dataDBFilename),
67            ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
68            ↪ IndexTreeType.Default);
69    }
70
71    /// <summary>
72    /// <para>
73    /// Creates the json storage using the specified links.
74    /// </para>
75    /// <para></para>
76    /// </summary>
77    /// <param name="links">
78    /// <para>The links.</para>
79    /// <para></para>
80    /// </param>
81    /// <returns>
82    /// <para>A default json storage of t link</para>
83    /// <para></para>
84    /// </returns>
85    public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links) => new
86    ↪ (links, BalancedVariantConverter);
87
88    /// <summary>
89    /// <para>
90    /// Imports the storage.
91    /// </para>
92    /// <para></para>
93    /// </summary>
94    /// <param name="storage">
95    /// <para>The storage.</para>
96    /// <para></para>
97    /// </param>
98    /// <param name="documentName">
99    /// <para>The document name.</para>
100    /// <para></para>
101    /// </param>
102    /// <param name="json">
103    /// <para>The json.</para>
104    /// <para></para>
105    /// </param>
106    /// <returns>
107    /// <para>The link</para>
108    /// <para></para>
109    /// </returns>
110    public TLink Import(IJsonStorage<TLink> storage, string documentName, byte[] json)
111    {
112        Utf8JsonReader utf8JsonReader = new(json);
113        JsonImporter<TLink> jsonImporter = new(storage);
114        CancellationTokenSource importCancellationTokenSource = new();
115        CancellationToken cancellationToken = importCancellationTokenSource.Token;
116        return jsonImporter.Import(documentName, ref utf8JsonReader, in cancellationToken);
117    }
118
119    /// <summary>
120    /// <para>
121    /// Exports the document link.
122    /// </para>
123    /// <para></para>
124    /// </summary>
125    /// <param name="documentLink">
126    /// <para>The document link.</para>
127    /// <para></para>
128    /// </param>

```

```

123     /// </param>
124     /// <param name="storage">
125     /// <para>The storage.</para>
126     /// <para></para>
127     /// </param>
128     /// <param name="stream">
129     /// <para>The stream.</para>
130     /// <para></para>
131     /// </param>
132     public void Export(TLink documentLink, IJsonStorage<TLink> storage, in MemoryStream
    ↪ stream)
133     {
134         Utf8JsonWriter writer = new(stream);
135         JsonExporter<TLink> jsonExporter = new(storage);
136         CancellationTokenSource exportCancellationTokenSource = new();
137         CancellationToken exportCancellationToken = exportCancellationTokenSource.Token;
138         jsonExporter.Export(documentLink, ref writer, in exportCancellationToken);
139         writer.Dispose();
140     }
141
142     /// <summary>
143     /// <para>
144     /// Tests that test.
145     /// </para>
146     /// <para></para>
147     /// </summary>
148     /// <param name="initialJson">
149     /// <para>The initial json.</para>
150     /// <para></para>
151     /// </param>
152     [Theory]
153     [InlineData("{}")]
154     [InlineData("\"stringValue\"")]
155     [InlineData("228")]
156     [InlineData("0.5")]
157     [InlineData("[]")]
158     [InlineData("true")]
159     [InlineData("false")]
160     [InlineData("null")]
161     [InlineData("{ \"string\": \"string\" }")]
162     [InlineData("{ \"null\": null }")]
163     [InlineData("{ \"boolean\": false }")]
164     [InlineData("{ \"boolean\": true }")]
165     [InlineData("{ \"array\": [] }")]
166     [InlineData("{ \"array\": [1] }")]
167     [InlineData("{ \"object\": {} }")]
168     [InlineData("{ \"number\": 1 }")]
169     [InlineData("{ \"decimal\": 0.5 }")]
170     [InlineData("[null]")]
171     [InlineData("[true]")]
172     [InlineData("[false]")]
173     [InlineData("[[]]")]
174     [InlineData("[[1]]")]
175     [InlineData("[[0.5]]")]
176     [InlineData("[{}]")]
177     [InlineData("[\"The Venus Project\"]")]
178     [InlineData("[{ \"title\": \"The Venus Project\" } ]")]
179     [InlineData("[1,2,3,4]")]
180     [InlineData("[-0.5, 0.5]")]
181     public void Test(string initialJson)
182     {
183         var links = CreateLinks();
184         BalancedVariantConverter = new(links);
185         var storage = CreateJsonStorage(links);
186         var json = Encoding.UTF8.GetBytes(initialJson);
187         var documentLink = Import(storage, "documentName", json);
188         MemoryStream stream = new();
189         Export(documentLink, storage, in stream);
190         string exportedJson = Encoding.UTF8.GetString(stream.ToArray());
191         stream.Dispose();
192         var minimizedInitialJson = Regex.Replace(initialJson,
    ↪ "(\\"(?:[^\\"\\\\]|\\\\\\\\|\\\\\\.|\\\\\\s+)*\\")\\\\\\s+", "$1");
193         Assert.Equal(minimizedInitialJson, exportedJson);
194     }
195 }
196 }

```

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

```

1  using Xunit;
2  using Platform.Data.Doublets.Memory.United.Generic;
3  using Platform.Data.Doublets.Memory;
4  using Platform.Memory;
5  using TLink = System.UInt32;
6  using Xunit.Abstractions;
7  using Platform.Collections.Stacks;
8  using Platform.Data.Doublets.Sequences.Walkers;
9  using System.Collections.Generic;
10 using Platform.Data.Doublets.Sequences.Converters;
11
12 namespace Platform.Data.Doublets.Json.Tests
13 {
14     /// <summary>
15     /// <para>
16     /// Represents the json storage tests.
17     /// </para>
18     /// <para></para>
19     /// </summary>
20     public class JsonStorageTests
21     {
22         /// <summary>
23         /// <para>
24         /// The output.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         private readonly ITestOutputHelper output;
29         /// <summary>
30         /// <para>
31         /// The balanced variant converter.
32         /// </para>
33         /// <para></para>
34         /// </summary>
35         public static BalancedVariantConverter<TLink> BalancedVariantConverter;
36
37         /// <summary>
38         /// <para>
39         /// Initializes a new <see cref="JsonStorageTests"/> instance.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         /// <param name="output">
44         /// <para>A output.</para>
45         /// <para></para>
46         /// </param>
47         public JsonStorageTests(ITestOutputHelper output)
48         {
49             this.output = output;
50         }
51
52         /// <summary>
53         /// <para>
54         /// Creates the links.
55         /// </para>
56         /// <para></para>
57         /// </summary>
58         /// <returns>
59         /// <para>A links of t link</para>
60         /// <para></para>
61         /// </returns>
62         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
        ↪ Platform.IO.TemporaryFile());
63
64         /// <summary>
65         /// <para>
66         /// Creates the links using the specified data db filename.
67         /// </para>
68         /// <para></para>
69         /// </summary>
70         /// <typeparam name="TLink">
71         /// <para>The link.</para>
72         /// <para></para>
73         /// </typeparam>
74         /// <param name="dataDBFilename">
75         /// <para>The data db filename.</para>
76         /// <para></para>
77         /// </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:
85             ↪ true);
86         return new UnitedMemoryLinks<TLink>(new
87             ↪ FileMappedResizableDirectMemory(dataDBFilename),
88             ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
89             ↪ IndexTreeType.Default);
90     }
91
92     /// <summary>
93     /// <para>
94     /// Creates the json storage.
95     /// </para>
96     /// <para></para>
97     /// </summary>
98     /// <returns>
99     /// <para>A default json storage of t link</para>
100    /// <para></para>
101    /// </returns>
102    public static DefaultJsonStorage<TLink> CreateJsonStorage()
103    {
104        var links = CreateLinks();
105        return CreateJsonStorage(links);
106    }
107
108    /// <summary>
109    /// <para>
110    /// Creates the json storage using the specified links.
111    /// </para>
112    /// <para></para>
113    /// </summary>
114    /// <param name="links">
115    /// <para>The links.</para>
116    /// <para></para>
117    /// </param>
118    /// <returns>
119    /// <para>A default json storage of t link</para>
120    /// <para></para>
121    /// </returns>
122    public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links)
123    {
124        BalancedVariantConverter = new(links);
125        return new DefaultJsonStorage<TLink>(links, BalancedVariantConverter);
126    }
127
128    /// <summary>
129    /// <para>
130    /// Tests that constructors test.
131    /// </para>
132    /// <para></para>
133    /// </summary>
134    [Fact]
135    public void ConstructorsTest() => CreateJsonStorage();
136
137    /// <summary>
138    /// <para>
139    /// Tests that create document test.
140    /// </para>
141    /// <para></para>
142    /// </summary>
143    [Fact]
144    public void CreateDocumentTest()
145    {
146        var defaultJsonStorage = CreateJsonStorage();
147        defaultJsonStorage.CreateDocument("documentName");
148    }
149
150    /// <summary>
151    /// <para>
152    /// Tests that get document test.
153    /// </para>
154    /// <para></para>
155    /// </summary>

```

```

152 [Fact]
153 public void GetDocumentTest()
154 {
155     var defaultJsonStorage = CreateJsonStorage();
156     var createdDocumentLink = defaultJsonStorage.CreateDocument("documentName");
157     var foundDocumentLink = defaultJsonStorage.GetDocumentOrDefault("documentName");
158     Assert.Equal(createdDocumentLink, foundDocumentLink);
159 }
160
161 /// <summary>
162 /// <para>
163 /// Tests that create object test.
164 /// </para>
165 /// <para></para>
166 /// </summary>
167 [Fact]
168 public void CreateObjectTest()
169 {
170     var defaultJsonStorage = CreateJsonStorage();
171     var object0 = defaultJsonStorage.CreateObjectValue();
172     var object1 = defaultJsonStorage.CreateObjectValue();
173     Assert.NotEqual(object0, object1);
174 }
175
176 /// <summary>
177 /// <para>
178 /// Tests that create string test.
179 /// </para>
180 /// <para></para>
181 /// </summary>
182 [Fact]
183 public void CreateStringTest()
184 {
185     var defaultJsonStorage = CreateJsonStorage();
186     defaultJsonStorage.CreateString("string");
187 }
188
189 /// <summary>
190 /// <para>
191 /// Tests that create member test.
192 /// </para>
193 /// <para></para>
194 /// </summary>
195 [Fact]
196 public void CreateMemberTest()
197 {
198     var defaultJsonStorage = CreateJsonStorage();
199     var document = defaultJsonStorage.CreateDocument("documentName");
200     defaultJsonStorage.AttachObject(document);
201     defaultJsonStorage.CreateMember("keyName");
202 }
203
204 /// <summary>
205 /// <para>
206 /// Tests that attach object value to document test.
207 /// </para>
208 /// <para></para>
209 /// </summary>
210 [Fact]
211 public void AttachObjectValueToDocumentTest()
212 {
213     var links = CreateLinks();
214     var defaultJsonStorage = CreateJsonStorage(links);
215     TLink document = defaultJsonStorage.CreateDocument("documentName");
216     TLink documentValueLink = defaultJsonStorage.AttachObject(document);
217     TLink createdObjectValue = links.GetTarget(documentValueLink);
218
219     TLink valueMarker = links.GetSource(createdObjectValue);
220     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
221
222     TLink createdObject = links.GetTarget(createdObjectValue);
223     TLink objectMarker = links.GetSource(createdObject);
224     Assert.Equal(objectMarker, defaultJsonStorage.ObjectMarker);
225
226     TLink foundDocumentValue = defaultJsonStorage.GetValueLink(document);
227     Assert.Equal(createdObjectValue, foundDocumentValue);
228 }
229

```

```

230 /// <summary>
231 /// <para>
232 /// Tests that attach string value to document test.
233 /// </para>
234 /// <para></para>
235 /// </summary>
236 [Fact]
237 public void AttachStringValueToDocumentTest()
238 {
239     var links = CreateLinks();
240     var defaultJsonStorage = CreateJsonStorage(links);
241     TLink document = defaultJsonStorage.CreateDocument("documentName");
242     TLink documentStringLink = defaultJsonStorage.AttachString(document, "stringName");
243     TLink createdStringValue = links.GetTarget(documentStringLink);
244
245     TLink valueMarker = links.GetSource(createdStringValue);
246     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
247
248     TLink createdString = links.GetTarget(createdStringValue);
249     TLink stringMarker = links.GetSource(createdString);
250     Assert.Equal(stringMarker, defaultJsonStorage.StringMarker);
251
252     TLink foundStringValue = defaultJsonStorage.GetValueLink(document);
253     Assert.Equal(createdStringValue, foundStringValue);
254 }
255
256 /// <summary>
257 /// <para>
258 /// Tests that attach number to document test.
259 /// </para>
260 /// <para></para>
261 /// </summary>
262 [Fact]
263 public void AttachNumberToDocumentTest()
264 {
265     var links = CreateLinks();
266     var defaultJsonStorage = CreateJsonStorage(links);
267     TLink document = defaultJsonStorage.CreateDocument("documentName");
268     TLink documentNumberLink = defaultJsonStorage.AttachNumber(document, 2021);
269     TLink createdNumberValue = links.GetTarget(documentNumberLink);
270
271     TLink valueMarker = links.GetSource(createdNumberValue);
272     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
273
274     TLink createdNumber = links.GetTarget(createdNumberValue);
275     TLink numberMarker = links.GetSource(createdNumber);
276     Assert.Equal(numberMarker, defaultJsonStorage.NumberMarker);
277
278     TLink foundNumberValue = defaultJsonStorage.GetValueLink(document);
279     Assert.Equal(createdNumberValue, foundNumberValue);
280 }
281
282 /// <summary>
283 /// <para>
284 /// Tests that attach true value to document test.
285 /// </para>
286 /// <para></para>
287 /// </summary>
288 [Fact]
289 public void AttachTrueValueToDocumentTest()
290 {
291     var links = CreateLinks();
292     var defaultJsonStorage = CreateJsonStorage(links);
293     TLink document = defaultJsonStorage.CreateDocument("documentName");
294
295     TLink documentTrueValueLink = defaultJsonStorage.AttachBoolean(document, true);
296     TLink createdTrueValue = links.GetTarget(documentTrueValueLink);
297
298     TLink valueMarker = links.GetSource(createdTrueValue);
299     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
300
301     TLink trueMarker = links.GetTarget(createdTrueValue);
302     Assert.Equal(trueMarker, defaultJsonStorage.TrueMarker);
303
304     TLink foundTrueValue = defaultJsonStorage.GetValueLink(document);
305     Assert.Equal(createdTrueValue, foundTrueValue);
306 }
307
308 /// <summary>

```



```

309 /// <para>
310 /// Tests that attach false value to document test.
311 /// </para>
312 /// <para></para>
313 /// </summary>
314 [Fact]
315 public void AttachFalseValueToDocumentTest()
316 {
317     var links = CreateLinks();
318     var defaultJsonStorage = CreateJsonStorage(links);
319     TLink document = defaultJsonStorage.CreateDocument("documentName");
320
321     TLink documentFalseValueLink = defaultJsonStorage.AttachBoolean(document, false);
322     TLink createdFalseValue = links.GetTarget(documentFalseValueLink);
323
324     TLink valueMarker = links.GetSource(createdFalseValue);
325     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
326
327     TLink falseMarker = links.GetTarget(createdFalseValue);
328     Assert.Equal(falseMarker, defaultJsonStorage.FalseMarker);
329
330     TLink foundFalseValue = defaultJsonStorage.GetValueLink(document);
331     Assert.Equal(createdFalseValue, foundFalseValue);
332 }
333
334 /// <summary>
335 /// <para>
336 /// Tests that attach null value to document test.
337 /// </para>
338 /// <para></para>
339 /// </summary>
340 [Fact]
341 public void AttachNullValueToDocumentTest()
342 {
343     var links = CreateLinks();
344     var defaultJsonStorage = CreateJsonStorage(links);
345     TLink document = defaultJsonStorage.CreateDocument("documentName");
346
347     TLink documentNullValueLink = defaultJsonStorage.AttachNull(document);
348     TLink createdNullValue = links.GetTarget(documentNullValueLink);
349
350     TLink valueMarker = links.GetSource(createdNullValue);
351     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
352
353     TLink nullMarker = links.GetTarget(createdNullValue);
354     Assert.Equal(nullMarker, defaultJsonStorage.NullMarker);
355
356     TLink foundNullValue = defaultJsonStorage.GetValueLink(document);
357     Assert.Equal(createdNullValue, foundNullValue);
358 }
359
360 /// <summary>
361 /// <para>
362 /// Tests that attach empty array value to document test.
363 /// </para>
364 /// <para></para>
365 /// </summary>
366 [Fact]
367 public void AttachEmptyArrayValueToDocumentTest()
368 {
369     var links = CreateLinks();
370     var defaultJsonStorage = CreateJsonStorage(links);
371     TLink document = defaultJsonStorage.CreateDocument("documentName");
372
373     TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, new
374         ↵ TLink[0]);
375     TLink createdArrayValue = links.GetTarget(documentArrayValueLink);
376     output.WriteLine(links.Format(createdArrayValue));
377
378     TLink valueMarker = links.GetSource(createdArrayValue);
379     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
380
381     TLink createdArrayLink = links.GetTarget(createdArrayValue);
382     TLink arrayMarker = links.GetSource(createdArrayLink);
383     Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
384
385     TLink createArrayContents = links.GetTarget(createdArrayLink);
386     Assert.Equal(createArrayContents, defaultJsonStorage.EmptyArrayMarker);

```

```

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

```

```

464     ILinks<TLink> links = CreateLinks();
465     var defaultJsonStorage = CreateJsonStorage(links);
466     TLink document = defaultJsonStorage.CreateDocument("documentName");
467     TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
468     TLink objectValueLink = links.GetTarget(documentObjectValueLink);
469     TLink objectFromGetObject = defaultJsonStorage.GetObject(objectValueLink);
470     Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
471 }
472
473 /// <summary>
474 /// <para>
475 /// Tests that attach string value to key.
476 /// </para>
477 /// <para></para>
478 /// </summary>
479 [Fact]
480 public void AttachStringValueToKey()
481 {
482     ILinks<TLink> links = CreateLinks();
483     var defaultJsonStorage = CreateJsonStorage(links);
484     TLink document = defaultJsonStorage.CreateDocument("documentName");
485     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
486     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
487     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
488     TLink memberStringValueLink = defaultJsonStorage.AttachString(memberLink,
489         ↪ "stringValue");
489     TLink stringValueLink = links.GetTarget(memberStringValueLink);
490     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
491     Assert.Equal(memberLink, objectMembersLinks[0]);
492     Assert.Equal(stringValueLink,
493         ↪ defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
493 }
494
495 /// <summary>
496 /// <para>
497 /// Tests that attach number value to key.
498 /// </para>
499 /// <para></para>
500 /// </summary>
501 [Fact]
502 public void AttachNumberValueToKey()
503 {
504     ILinks<TLink> links = CreateLinks();
505     var defaultJsonStorage = CreateJsonStorage(links);
506     TLink document = defaultJsonStorage.CreateDocument("documentName");
507     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
508     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
509     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
510     TLink memberNumberValueLink = defaultJsonStorage.AttachNumber(memberLink, 123);
511     TLink numberValueLink = links.GetTarget(memberNumberValueLink);
512     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
513     Assert.Equal(memberLink, objectMembersLinks[0]);
514     Assert.Equal(numberValueLink,
515         ↪ defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
515 }
516
517 /// <summary>
518 /// <para>
519 /// Tests that attach object value to key.
520 /// </para>
521 /// <para></para>
522 /// </summary>
523 [Fact]
524 public void AttachObjectValueToKey()
525 {
526     ILinks<TLink> links = CreateLinks();
527     var defaultJsonStorage = CreateJsonStorage(links);
528     TLink document = defaultJsonStorage.CreateDocument("documentName");
529     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
530     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
531     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
532     TLink memberObjectValueLink = defaultJsonStorage.AttachObject(memberLink);
533     TLink objectValueLink = links.GetTarget(memberObjectValueLink);
534     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
535     Assert.Equal(memberLink, objectMembersLinks[0]);
536     Assert.Equal(objectValueLink,
537         ↪ defaultJsonStorage.GetValueLink(objectMembersLinks[0]));

```

```

537 }
538
539 /// <summary>
540 /// <para>
541 /// Tests that attach array value to key.
542 /// </para>
543 /// <para></para>
544 /// </summary>
545 [Fact]
546 public void AttachArrayValueToKey()
547 {
548     ILinks<TLink> links = CreateLinks();
549     var defaultJsonStorage = CreateJsonStorage(links);
550     TLink document = defaultJsonStorage.CreateDocument("documentName");
551     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
552     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
553     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
554     TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
555     TLink[] array = { arrayElement, arrayElement, arrayElement };
556     TLink memberArrayValueLink = defaultJsonStorage.AttachArray(memberLink, array);
557     TLink arrayValueLink = links.GetTarget(memberArrayValueLink);
558     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
559     Assert.Equal(memberLink, objectMembersLinks[0]);
560     Assert.Equal(arrayValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
561 }
562
563 /// <summary>
564 /// <para>
565 /// Tests that attach true value to key.
566 /// </para>
567 /// <para></para>
568 /// </summary>
569 [Fact]
570 public void AttachTrueValueToKey()
571 {
572     ILinks<TLink> links = CreateLinks();
573     var defaultJsonStorage = CreateJsonStorage(links);
574     TLink document = defaultJsonStorage.CreateDocument("documentName");
575     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
576     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
577     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
578     TLink memberTrueValueLink = defaultJsonStorage.AttachBoolean(memberLink, true);
579     TLink trueValueLink = links.GetTarget(memberTrueValueLink);
580     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
581     Assert.Equal(memberLink, objectMembersLinks[0]);
582     Assert.Equal(trueValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
583 }
584
585 /// <summary>
586 /// <para>
587 /// Tests that attach false value to key.
588 /// </para>
589 /// <para></para>
590 /// </summary>
591 [Fact]
592 public void AttachFalseValueToKey()
593 {
594     ILinks<TLink> links = CreateLinks();
595     var defaultJsonStorage = CreateJsonStorage(links);
596     TLink document = defaultJsonStorage.CreateDocument("documentName");
597     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
598     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
599     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
600     TLink memberFalseValueLink = defaultJsonStorage.AttachBoolean(memberLink, false);
601     TLink falseValueLink = links.GetTarget(memberFalseValueLink);
602     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
603     Assert.Equal(memberLink, objectMembersLinks[0]);
604     Assert.Equal(falseValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
605 }
606
607 /// <summary>
608 /// <para>
609 /// Tests that attach null value to key.
610 /// </para>
611 /// <para></para>
612 /// </summary>
613 [Fact]
614 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

./csharp/Platform.Data.Doublets.Json.Tests/JsonImportAndExportTests.cs, 34
./csharp/Platform.Data.Doublets.Json.Tests/JsonStorageTests.cs, 36
./csharp/Platform.Data.Doublets.Json/DefaultJsonStorage.cs, 1
./csharp/Platform.Data.Doublets.Json/IJsonStorage.cs, 15
./csharp/Platform.Data.Doublets.Json/JsonArrayElementCriterionMatcher.cs, 23
./csharp/Platform.Data.Doublets.Json/JsonExporter.cs, 24
./csharp/Platform.Data.Doublets.Json/JsonExporterCli.cs, 28
./csharp/Platform.Data.Doublets.Json/JsonImporter.cs, 30
./csharp/Platform.Data.Doublets.Json/JsonImporterCli.cs, 33