

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

```

```

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

```

```

229     /// </summary>
230     public TLink ArrayMarker { get; }
231     /// <summary>
232     /// <para>
233     /// Gets the empty array marker value.
234     /// </para>
235     /// <para></para>
236     /// </summary>
237     public TLink EmptyArrayMarker { get; }
238     /// <summary>
239     /// <para>
240     /// Gets the true marker value.
241     /// </para>
242     /// <para></para>
243     /// </summary>
244     public TLink TrueMarker { get; }
245     /// <summary>
246     /// <para>
247     /// Gets the false marker value.
248     /// </para>
249     /// <para></para>
250     /// </summary>
251     public TLink FalseMarker { get; }
252     /// <summary>
253     /// <para>
254     /// Gets the null marker value.
255     /// </para>
256     /// <para></para>
257     /// </summary>
258     public TLink NullMarker { get; }
259
260     /// <summary>
261     /// <para>
262     /// Initializes a new <see cref="DefaultJsonStorage"/> instance.
263     /// </para>
264     /// <para></para>
265     /// </summary>
266     /// <param name="links">
267     /// <para>A links.</para>
268     /// <para></para>
269     /// </param>
270     /// <param name="listToSequenceConverter">
271     /// <para>A list to sequence converter.</para>
272     /// <para></para>
273     /// </param>
274     public DefaultJsonStorage(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
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 /// </summary>
332 /// <param name="content">
333 /// <para>The content.</para>
334 /// </param>
335 /// <returns>
336 /// <para>The link</para>
337 /// </returns>
338 public TLink CreateString(string content)
339 {
340     var @string = GetStringSequence(content);
341     return Links.GetOrCreate(StringMarker, @string);
342 }
343
344 /// <summary>
345 /// <para>
346 /// Creates the string value using the specified content.
347 /// </para>
348 /// </summary>
349 /// <param name="content">
350 /// <para>The content.</para>
351 /// </param>
352 /// <returns>
353 /// <para>The link</para>
354 /// </returns>
355 public TLink CreateStringValue(string content)
356 {
357     var @string = CreateString(content);
358     return CreateValue(@string);
359 }
360
361 /// <summary>
362 /// <para>
363 /// Creates the number using the specified number.
364 /// </para>
365 /// </summary>
366 /// <param name="number">
367 /// <para>The number.</para>
368 /// </param>
369 /// </summary>
370

```

```

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    private TLink GetStringSequence(string content) => content == "" ? EmptyStringMarker :
833        ↪ StringToUnicodeSequenceConverter.Convert(content);
834
835    /// <summary>
836    /// <para>
837    /// Gets the string using the specified string value.

```

```

837     /// </para>
838     /// <para></para>
839     /// </summary>
840     /// <param name="stringValue">
841     /// <para>The string value.</para>
842     /// <para></para>
843     /// </param>
844     /// <exception cref="Exception">
845     /// <para>The passed link does not contain a string.</para>
846     /// <para></para>
847     /// </exception>
848     /// <returns>
849     /// <para>The string</para>
850     /// <para></para>
851     /// </returns>
852     public string GetString(TLink stringValue)
853     {
854         var current = stringValue;
855         TLink source;
856         for (int i = 0; i < 3; i++)
857         {
858             source = Links.GetSource(current);
859             if (EqualityComparer.Equals(source, StringMarker))
860             {
861                 var sequence = Links.GetTarget(current);
862                 var isEmpty = EqualityComparer.Equals(sequence, EmptyStringMarker);
863                 return isEmpty ? "" : UnicodeSequenceToStringConverter.Convert(sequence);
864             }
865             current = Links.GetTarget(current);
866         }
867         throw new Exception("The passed link does not contain a string.");
868     }
869
870     /// <summary>
871     /// <para>
872     /// Gets the number using the specified value link.
873     /// </para>
874     /// <para></para>
875     /// </summary>
876     /// <param name="valueLink">
877     /// <para>The value link.</para>
878     /// <para></para>
879     /// </param>
880     /// <exception cref="Exception">
881     /// <para>The passed link does not contain a number.</para>
882     /// <para></para>
883     /// </exception>
884     /// <returns>
885     /// <para>The decimal</para>
886     /// <para></para>
887     /// </returns>
888     public decimal GetNumber(TLink valueLink)
889     {
890         var current = valueLink;
891         TLink source;
892         TLink target;
893         for (int i = 0; i < 3; i++)
894         {
895             source = Links.GetSource(current);
896             target = Links.GetTarget(current);
897             if (EqualityComparer.Equals(source, NumberMarker))
898             {
899                 return RationalToDecimalConverter.Convert(target);
900             }
901             current = target;
902         }
903         throw new Exception("The passed link does not contain a number.");
904     }
905
906
907     /// <summary>
908     /// <para>
909     /// Gets the object using the specified object value link.
910     /// </para>
911     /// <para></para>
912     /// </summary>
913     /// <param name="objectValueLink">
914     /// <para>The object value link.</para>

```

```

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

```

```

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

```

```

1071     /// <para></para>
1072     /// </summary>
1073     /// <param name="@object">
1074     /// <para>The object.</para>
1075     /// <para></para>
1076     /// </param>
1077     /// <returns>
1078     /// <para>The members.</para>
1079     /// <para></para>
1080     /// </returns>
1081     public List<TLink> GetMembersLinks(TLink @object)
1082     {
1083         Link<TLink> query = new(index: Any, source: @object, target: Any);
1084         List<TLink> members = new();
1085         Links.Each(objectMemberLink =>
1086         {
1087             var memberLink = Links.GetTarget(objectMemberLink);
1088             var memberMarker = Links.GetSource(memberLink);
1089             if (EqualityComparer.Equals(memberMarker, MemberMarker))
1090             {
1091                 members.Add(Links.GetIndex(objectMemberLink));
1092             }
1093             return Links.Constants.Continue;
1094         }, query);
1095         return members;
1096     }
1097 }
1098 }

```

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
37         /// <summary>
38         /// <para>
39         /// Initializes a new <see cref="JsonArrayElementCriterionMatcher" /> instance.
40         /// </para>
41         /// <para></para>
42         /// </summary>
43         /// <param name="storage">
44         /// <para>A storage.</para>
45         /// <para></para>
46         /// </param>
47         public JsonArrayElementCriterionMatcher(IJsonStorage<TLink> storage) => Storage =
48             ↪ storage;
49
50         /// <summary>
51         /// <para>
52         /// Determines whether this instance is matched.
53         /// </para>
54         /// <para></para>
55         /// </summary>
56         /// <param name="link">
57         /// <para>The link.</para>
58         /// <para></para>
59         /// </param>
60         /// <returns>
61         /// <para>The bool</para>
62         /// <para></para>
63         /// </returns>
64         public bool IsMatched(TLink link) =>
65             ↪ EqualityComparer<TLink>.Default.Equals(Storage.Links.GetSource(link),
66             ↪ Storage.ValueMarker);
67     }
68 }

```

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 =
34             ↳ EqualityComparer<TLink>.Default;
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         private bool IsElement(TLink link)
48         {
49             var marker = Storage.Links.GetSource(link);
50             return EqualityComparer.Equals(marker, Storage.ValueMarker);
51         }
52
53         private void WriteStringValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
54             ↳ utf8JsonWriter.WriteStringValue(Storage.GetString(valueLink));
55
56         private void WriteString(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
57             ↳ valueLink) => utf8JsonWriter.WriteString(parent, Storage.GetString(valueLink));
58
59         private void WriteNumberValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
60             ↳ utf8JsonWriter.WriteNumberValue(Storage.GetNumber(valueLink));
61
62         private void WriteNumber(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
63             ↳ valueLink) => utf8JsonWriter.WriteNumber(parent, Storage.GetNumber(valueLink));
64
65         private void Write(ref Utf8JsonWriter utf8JsonWriter, string parent, TLink valueLink,
66             ↳ Cancellation_token cancellation_token)
67         {
68             if (cancellation_token.IsCancellationRequested)
69             {
70                 return;
71             }
72             var valueMarker = Storage.GetValueMarker(valueLink);
73             if (EqualityComparer.Equals(valueMarker, Storage.ObjectMarker))
74             {
75                 utf8JsonWriter.WriteStartObject(parent);
76                 var membersLinks = Storage.GetMembersLinks(Storage.GetObject(valueLink));
77                 foreach (var memberLink in membersLinks)
```



```

73     {
74         if (cancellationToken.IsCancellationRequested)
75         {
76             return;
77         }
78         Write(ref utf8JsonWriter, Storage.GetString(memberLink),
79             ↪ Storage.GetValueLink(memberLink), cancellationToken);
80     }
81     utf8JsonWriter.WriteEndObject();
82 }
83 else if (EqualityComparer.Equals(valueMarker, Storage.ArrayMarker))
84 {
85     var array = Storage.GetArray(valueLink);
86     var sequence = Storage.GetArraySequence(array);
87     utf8JsonWriter.WriteStartArray(parent);
88     if (!EqualityComparer.Equals(sequence, Storage.EmptyArrayMarker))
89     {
90         RightSequenceWalker<TLink> rightSequenceWalker = new(Storage.Links, new
91             ↪ DefaultStack<TLink>(), IsElement);
92         var elements = rightSequenceWalker.Walk(sequence);
93         foreach (var element in elements)
94         {
95             if (cancellationToken.IsCancellationRequested)
96             {
97                 return;
98             }
99             Write(ref utf8JsonWriter, element, in cancellationToken);
100         }
101     }
102     utf8JsonWriter.WriteEndArray();
103 }
104 else if (EqualityComparer.Equals(valueMarker, Storage.StringMarker))
105 {
106     WriteString(in utf8JsonWriter, parent, valueLink);
107 }
108 else if (EqualityComparer.Equals(valueMarker, Storage.NumberMarker))
109 {
110     WriteNumber(in utf8JsonWriter, parent, valueLink);
111 }
112 else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
113 {
114     utf8JsonWriter.WriteBoolean(parent, true);
115 }
116 else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
117 {
118     utf8JsonWriter.WriteBoolean(parent, false);
119 }
120 else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
121 {
122     utf8JsonWriter.WriteNull(parent);
123 }
124 }
125
126 private void Write(ref Utf8JsonWriter utf8JsonWriter, TLink valueLink, in
127     ↪ CancellationToken cancellationToken)
128 {
129     if (cancellationToken.IsCancellationRequested)
130     {
131         return;
132     }
133     var valueMarker = Storage.GetValueMarker(valueLink);
134     if (EqualityComparer.Equals(valueMarker, Storage.ObjectMarker))
135     {
136         utf8JsonWriter.WriteStartObject();
137         var membersLinks = Storage.GetMembersLinks(Storage.GetObject(valueLink));
138         foreach (var memberLink in membersLinks)
139         {
140             if (cancellationToken.IsCancellationRequested)
141             {
142                 return;
143             }
144             Write(ref utf8JsonWriter, Storage.GetString(memberLink),
145                 ↪ Storage.GetValueLink(memberLink), cancellationToken);
146         }
147         utf8JsonWriter.WriteEndObject();
148     }
149     else if (EqualityComparer.Equals(valueMarker, Storage.ArrayMarker))
150     {

```

```

147     var array = Storage.GetArray(valueLink);
148     var sequence = Storage.GetArraySequence(array);
149     utf8JsonWriter.WriteStartArray();
150     if (!EqualityComparer.Equals(sequence, Storage.EmptyArrayMarker))
151     {
152         RightSequenceWalker<TLink> rightSequenceWalker = new(Storage.Links, new
            ↳ DefaultStack<TLink>(), IsElement);
153         var elements = rightSequenceWalker.Walk(sequence);
154         foreach (var element in elements)
155         {
156             if (cancellationToken.IsCancellationRequested)
157             {
158                 return;
159             }
160             Write(ref utf8JsonWriter, element, in cancellationToken);
161         }
162         utf8JsonWriter.WriteEndArray();
163     }
164     else if (EqualityComparer.Equals(valueMarker, Storage.StringMarker))
165     {
166         WriteStringValue(in utf8JsonWriter, valueLink);
167     }
168     else if (EqualityComparer.Equals(valueMarker, Storage.NumberMarker))
169     {
170         WriteNumberValue(in utf8JsonWriter, valueLink);
171     }
172     else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
173     {
174         utf8JsonWriter.WriteBooleanValue(true);
175     }
176     else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
177     {
178         utf8JsonWriter.WriteBooleanValue(false);
179     }
180     else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
181     {
182         utf8JsonWriter.WriteNullValue();
183     }
184 }
185
186 /// <summary>
187 /// <para>
188 /// Exports the document.
189 /// </para>
190 /// <para></para>
191 /// </summary>
192 /// <param name="document">
193 /// <para>The document.</para>
194 /// <para></para>
195 /// </param>
196 /// <param name="utf8JsonWriter">
197 /// <para>The utf json writer.</para>
198 /// <para></para>
199 /// </param>
200 /// <param name="cancellationToken">
201 /// <para>The cancellation token.</para>
202 /// <para></para>
203 /// </param>
204 /// <exception cref="Exception">
205 /// <para>No document with this name exists</para>
206 /// <para></para>
207 /// </exception>
208
209 public void Export(TLink document, ref Utf8JsonWriter utf8JsonWriter, in
    ↳ Cancellation token cancellationToken)
210 {
211     if (EqualityComparer.Equals(document, default))
212     {
213         throw new Exception("No document with this name exists");
214     }
215     var valueLink = Storage.GetValueLink(document);
216     Write(ref utf8JsonWriter, valueLink, in cancellationToken);
217     utf8JsonWriter.Flush();
218 }
219
220 /// <summary>
221 /// <para>
222 /// Exports the document name.

```

```

223     /// </para>
224     /// <para></para>
225     /// </summary>
226     /// <param name="documentName">
227     /// <para>The document name.</para>
228     /// <para></para>
229     /// </param>
230     /// <param name="utf8JsonWriter">
231     /// <para>The utf json writer.</para>
232     /// <para></para>
233     /// </param>
234     /// <param name="cancellationToken">
235     /// <para>The cancellation token.</para>
236     /// <para></para>
237     /// </param>
238     public void Export(string documentName, Utf8JsonWriter utf8JsonWriter, CancellationToken
        ↪ cancellationToken) => Export(Storage.GetDocumentOrDefault(documentName), ref
        ↪ utf8JsonWriter, in cancellationToken);
239 }
240 }

```

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
        ↪ path", args);
38             var jsonFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "JSON file
        ↪ path", args);
39             var defaultDocumentName = Path.GetFileNameWithoutExtension(jsonFilePath);
40             var documentName = ConsoleHelpers.GetOrReadArgument(argumentIndex, $"Document name
        ↪ (default: {defaultDocumentName})", args);
41             if (string.IsNullOrEmpty(documentName))
42             {
43                 documentName = defaultDocumentName;
44             }
45             if (!File.Exists(linksFilePath))
46             {
47                 Console.WriteLine($"${linksFilePath} file does not exist.");
48             }
49             using FileStream jsonFileStream = new(jsonFilePath, FileMode.Append);
50             JsonSerializerOptions utf8JsonWriterOptions = new()
51             {
52                 Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping,
53                 Indented = true
54             };

```

```

55     Utf8JsonWriter utf8JsonWriter = new(jsonFileStream, utf8JsonWriterOptions);
56     var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
    ↪ true);
57     using UnitedMemoryLinks<TLink> memoryAdapter = new (new
    ↪ FileMappedResizableDirectMemory(linksFilePath),
    ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
    ↪ IndexTreeType.Default);
58     var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
59     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
60     var storage = new DefaultJsonStorage<TLink>(links, balancedVariantConverter);
61     var exporter = new JsonExporter<TLink>(storage);
62     var document = storage.GetDocumentOrDefault(documentName);
63     if (storage.EqualityComparer.Equals(document, default))
64     {
65         Console.WriteLine("No document with this name.");
66     }
67     using ConsoleCancellation cancellation = new ();
68     var cancellationToken = cancellation.Token;
69     Console.WriteLine("Press CTRL+C to stop.");
70     try
71     {
72         exporter.Export(document, ref utf8JsonWriter, in cancellationToken);
73     }
74     catch (Exception exception)
75     {
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 =
    ↪ EqualityComparer<TLink>.Default;
32         /// <summary>
33         /// <para>
34         /// The parents.
35         /// </para>
36         /// <para></para>
37         /// </summary>
38         public readonly Stack<TLink> Parents = new ();
39         /// <summary>
40         /// <para>

```

```

/// Initializes a new <see cref="JsonImporter"/> instance.
/// </para>
/// </summary>
/// <param name="storage">
/// <para>A storage.</para>
/// </param>
public JsonImporter(IJsonStorage<TLink> storage) => Storage = storage;

    private void PopIfParentIsMember()
{
    var parent = Parents.Peek();
    var parentMarker = Storage.GetValueMarker(parent);
    if (EqualityComparer.Equals(parentMarker, Storage.MemberMarker))
    {
        Parents.Pop();
    }
}

/// <summary>
/// <para>
/// Imports the document name.
/// </para>
/// </summary>
/// <param name="documentName">
/// <para>The document name.</para>
/// </param>
/// <param name="utf8JsonReader">
/// <para>The utf json reader.</para>
/// </param>
/// <param name="cancellationToken">
/// <para>The cancellation token.</para>
/// </param>
/// <exception cref="Exception">
/// <para>The document with the specified name already exists.</para>
/// </exception>
/// <returns>
/// <para>The document.</para>
/// </returns>
public TLink Import(string documentName, ref Utf8JsonReader utf8JsonReader, in
    CancellationToken cancellationToken)
{
    Parents.Clear();
    if (!EqualityComparer.Equals(Storage.GetDocumentOrDefault(documentName), default))
    {
        throw new Exception("The document with the specified name already exists.");
    }
    var document = Storage.CreateDocument(documentName);
    Parents.Push(document);
    TLink parent;
    TLink parentMarker;
    JsonTokenType tokenType;
    TLink value;
    TLink newParentArray;
    while (utf8JsonReader.Read())
    {
        cancellationToken.ThrowIfCancellationRequested();
        parent = Parents.Peek();
        parentMarker = Storage.GetValueMarker(parent);
        tokenType = utf8JsonReader.TokenType;
        if (utf8JsonReader.TokenType == JsonTokenType.PropertyName)
        {
            var @object = Storage.GetObject(parent);
            var property = utf8JsonReader.GetString();
            Parents.Push(Storage.AttachMemberToObject(@object, property));
        }
        switch (tokenType)
        {
            case JsonTokenType.StartObject:
            {
                value = Storage.CreateObjectValue();
            }
        }
    }
}

```

```

118         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
119         {
120             Parents.Pop();
121             newParentArray = Storage.AppendArrayValue(parent, value);
122             Parents.Push(newParentArray);
123             Parents.Push(value);
124         }
125         else
126         {
127             var @object = Storage.Attach(parent, value);
128             Parents.Push(@object);
129         }
130         break;
131     }
132     case JsonTokenType.EndObject:
133         Parents.Pop();
134         break;
135     case JsonTokenType.StartArray:
136         value = Storage.CreateArrayValue(Array.Empty<TLink>());
137         Parents.Push(value);
138         break;
139     case JsonTokenType.EndArray:
140     {
141         var arrayValue = Parents.Pop();
142         parent = Parents.Peek();
143         parentMarker = Storage.GetValueMarker(parent);
144         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
145         {
146             Parents.Pop();
147             newParentArray = Storage.AppendArrayValue(parent, arrayValue);
148             Parents.Push(newParentArray);
149         }
150         Storage.Attach(parent, arrayValue);
151         break;
152     }
153     case JsonTokenType.String:
154     {
155         var @string = utf8JsonReader.GetString();
156         value = Storage.CreateStringValue(@string);
157         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
158         {
159             Parents.Pop();
160             newParentArray = Storage.AppendArrayValue(parent, value);
161             Parents.Push(newParentArray);
162         }
163         else
164         {
165             Storage.Attach(parent, value);
166         }
167         break;
168     }
169     case JsonTokenType.Number:
170     {
171         value = Storage.CreateNumberValue(utf8JsonReader.GetDecimal());
172         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
173         {
174             Parents.Pop();
175             newParentArray = Storage.AppendArrayValue(parent, value);
176             Parents.Push(newParentArray);
177         }
178         else
179         {
180             Storage.Attach(parent, value);
181         }
182         break;
183     }
184     case JsonTokenType.True:
185     {
186         value = Storage.CreateBooleanValue(true);
187         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
188         {
189             Parents.Pop();
190             newParentArray = Storage.AppendArrayValue(parent, value);
191             Parents.Push(newParentArray);
192         }
193         else
194         {
195             Storage.Attach(parent, value);
196         }

```

```

197         break;
198     }
199     case JsonTokenType.False:
200     {
201         value = Storage.CreateBooleanValue(false);
202         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
203         {
204             Parents.Pop();
205             newParentArray = Storage.AppendArrayValue(parent, value);
206             Parents.Push(newParentArray);
207         }
208         else
209         {
210             Storage.Attach(parent, value);
211         }
212         break;
213     }
214     case JsonTokenType.Null:
215     {
216         value = Storage.CreateNullValue();
217         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
218         {
219             Parents.Pop();
220             newParentArray = Storage.AppendArrayValue(parent, value);
221             Parents.Push(newParentArray);
222         }
223         else
224         {
225             Storage.Attach(parent, value);
226         }
227         break;
228     }
229 }
230 if (tokenType != JsonTokenType.PropertyName && tokenType !=
    ↳ JsonTokenType.StartObject && tokenType != JsonTokenType.StartArray)
231 {
232     PopIfParentIsMember();
233 }
234 }
235 return document;
236 }
237 }
238 }

```

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,
    ↪ unitedMemoryLinks, linksConstants, indexTreeType);
58     var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
59     BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
60     DefaultJsonStorage<TLink> storage = new(links, balancedVariantConverter);
61     JsonImporter<TLink> importer = new(storage);
62     using ConsoleCancellation cancellation = new();
63     var cancellationToken = cancellation.Token;
64     Console.WriteLine("Press CTRL+C to stop.");
65     try
66     {
67         importer.Import(documentName, ref utf8JsonReader, in cancellationToken);
68     }
69     catch (Exception exception)
70     {
71         Console.WriteLine(exception);
72         return;
73     }
74     Console.WriteLine("Import completed successfully.");
75 }
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>

```

```

104 /// </returns>
105 public TLink Import(IJsonStorage<TLink> storage, string documentName, byte[] json)
106 {
107     Utf8JsonReader utf8JsonReader = new(json);
108     JsonImporter<TLink> jsonImporter = new(storage);
109     CancellationTokenSource importCancellationTokenSource = new();
110     CancellationToken cancellationToken = importCancellationTokenSource.Token;
111     return jsonImporter.Import(documentName, ref utf8JsonReader, in cancellationToken);
112 }
113
114 /// <summary>
115 /// <para>
116 /// Exports the document link.
117 /// </para>
118 /// <para></para>
119 /// </summary>
120 /// <param name="documentLink">
121 /// <para>The document link.</para>
122 /// <para></para>
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,
193         ↪ "(\\"(?:[^\\"\\\\]|\\\\\\\\.)*\\")\\\\s+", "$1");
194     Assert.Equal(minimizedInitialJson, exportedJson);
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         private readonly ITestOutputHelper output;
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         /// Initializes a new <see cref="JsonStorageTests"/> instance.
34         /// </para>
35         /// <para></para>
36         /// </summary>
37         /// <param name="output">
38         /// <para>A output.</para>
39         /// <para></para>
40         /// </param>
41         public JsonStorageTests(ITestOutputHelper output)
42         {
43             this.output = output;
44         }
45
46         /// <summary>
47         /// <para>
48         /// Creates the links.
49         /// </para>
50         /// <para></para>
51         /// </summary>
52         /// <returns>
53         /// <para>A links of t link</para>
54         /// <para></para>
55         /// </returns>
56         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
57             ↪ Platform.IO.TemporaryFile());
58
59         /// <summary>
60         /// <para>

```

```

60     /// Creates the links using the specified data db filename.
61     /// </para>
62     /// <para></para>
63     /// </summary>
64     /// <typeparam name="TLink">
65     /// <para>The link.</para>
66     /// <para></para>
67     /// </typeparam>
68     /// <param name="dataDBFilename">
69     /// <para>The data db filename.</para>
70     /// <para></para>
71     /// </param>
72     /// <returns>
73     /// <para>A links of t link</para>
74     /// <para></para>
75     /// </returns>
76     public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
77     {
78         var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
79             ↪ true);
80         return new UnitedMemoryLinks<TLink>(new
81             ↪ FileMappedResizableDirectMemory(dataDBFilename),
82             ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
83             ↪ IndexTreeType.Default);
84     }
85     /// <summary>
86     /// <para>
87     /// Creates the json storage.
88     /// </para>
89     /// <para></para>
90     /// </summary>
91     /// <returns>
92     /// <para>A default json storage of t link</para>
93     /// <para></para>
94     /// </returns>
95     public static DefaultJsonStorage<TLink> CreateJsonStorage()
96     {
97         var links = CreateLinks();
98         return CreateJsonStorage(links);
99     }
100     /// <summary>
101     /// <para>
102     /// Creates the json storage using the specified links.
103     /// </para>
104     /// <para></para>
105     /// </summary>
106     /// <param name="links">
107     /// <para>The links.</para>
108     /// <para></para>
109     /// </param>
110     /// <returns>
111     /// <para>A default json storage of t link</para>
112     /// <para></para>
113     /// </returns>
114     public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links)
115     {
116         BalancedVariantConverter = new(links);
117         return new DefaultJsonStorage<TLink>(links, BalancedVariantConverter);
118     }
119     /// <summary>
120     /// <para>
121     /// Tests that constructors test.
122     /// </para>
123     /// <para></para>
124     /// </summary>
125     [Fact]
126     public void ConstructorsTest() => CreateJsonStorage();
127     /// <summary>
128     /// <para>
129     /// Tests that create document test.
130     /// </para>
131     /// <para></para>
132     /// </summary>
133     [Fact]

```

```

134 public void CreateDocumentTest()
135 {
136     var defaultJsonStorage = CreateJsonStorage();
137     defaultJsonStorage.CreateDocument("documentName");
138 }
139
140 /// <summary>
141 /// <para>
142 /// Tests that get document test.
143 /// </para>
144 /// <para></para>
145 /// </summary>
146 [Fact]
147 public void GetDocumentTest()
148 {
149     var defaultJsonStorage = CreateJsonStorage();
150     var createdDocumentLink = defaultJsonStorage.CreateDocument("documentName");
151     var foundDocumentLink = defaultJsonStorage.GetDocumentOrDefault("documentName");
152     Assert.Equal(createdDocumentLink, foundDocumentLink);
153 }
154
155 /// <summary>
156 /// <para>
157 /// Tests that create object test.
158 /// </para>
159 /// <para></para>
160 /// </summary>
161 [Fact]
162 public void CreateObjectTest()
163 {
164     var defaultJsonStorage = CreateJsonStorage();
165     var object0 = defaultJsonStorage.CreateObjectValue();
166     var object1 = defaultJsonStorage.CreateObjectValue();
167     Assert.NotEqual(object0, object1);
168 }
169
170 /// <summary>
171 /// <para>
172 /// Tests that create string test.
173 /// </para>
174 /// <para></para>
175 /// </summary>
176 [Fact]
177 public void CreateStringTest()
178 {
179     var defaultJsonStorage = CreateJsonStorage();
180     defaultJsonStorage.CreateString("string");
181 }
182
183 /// <summary>
184 /// <para>
185 /// Tests that create member test.
186 /// </para>
187 /// <para></para>
188 /// </summary>
189 [Fact]
190 public void CreateMemberTest()
191 {
192     var defaultJsonStorage = CreateJsonStorage();
193     var document = defaultJsonStorage.CreateDocument("documentName");
194     defaultJsonStorage.AttachObject(document);
195     defaultJsonStorage.CreateMember("keyName");
196 }
197
198 /// <summary>
199 /// <para>
200 /// Tests that attach object value to document test.
201 /// </para>
202 /// <para></para>
203 /// </summary>
204 [Fact]
205 public void AttachObjectValueToDocumentTest()
206 {
207     var links = CreateLinks();
208     var defaultJsonStorage = CreateJsonStorage(links);
209     TLink document = defaultJsonStorage.CreateDocument("documentName");
210     TLink documentValueLink = defaultJsonStorage.AttachObject(document);
211     TLink createdObjectValue = links.GetTarget(documentValueLink);

```

```

212
213     TLink valueMarker = links.GetSource(createdObjectValue);
214     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
215
216     TLink createdObject = links.GetTarget(createdObjectValue);
217     TLink objectMarker = links.GetSource(createdObject);
218     Assert.Equal(objectMarker, defaultJsonStorage.ObjectMarker);
219
220     TLink foundDocumentValue = defaultJsonStorage.GetValueLink(document);
221     Assert.Equal(createdObjectValue, foundDocumentValue);
222 }
223
224 /// <summary>
225 /// <para>
226 /// Tests that attach string value to document test.
227 /// </para>
228 /// <para></para>
229 /// </summary>
230 [Fact]
231 public void AttachStringValueToDocumentTest()
232 {
233     var links = CreateLinks();
234     var defaultJsonStorage = CreateJsonStorage(links);
235     TLink document = defaultJsonStorage.CreateDocument("documentName");
236     TLink documentStringLink = defaultJsonStorage.AttachString(document, "stringName");
237     TLink createdStringValue = links.GetTarget(documentStringLink);
238
239     TLink valueMarker = links.GetSource(createdStringValue);
240     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
241
242     TLink createdString = links.GetTarget(createdStringValue);
243     TLink stringMarker = links.GetSource(createdString);
244     Assert.Equal(stringMarker, defaultJsonStorage.StringMarker);
245
246     TLink foundStringValue = defaultJsonStorage.GetValueLink(document);
247     Assert.Equal(createdStringValue, foundStringValue);
248 }
249
250 /// <summary>
251 /// <para>
252 /// Tests that attach number to document test.
253 /// </para>
254 /// <para></para>
255 /// </summary>
256 [Fact]
257 public void AttachNumberToDocumentTest()
258 {
259     var links = CreateLinks();
260     var defaultJsonStorage = CreateJsonStorage(links);
261     TLink document = defaultJsonStorage.CreateDocument("documentName");
262     TLink documentNumberLink = defaultJsonStorage.AttachNumber(document, 2021);
263     TLink createdNumberValue = links.GetTarget(documentNumberLink);
264
265     TLink valueMarker = links.GetSource(createdNumberValue);
266     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
267
268     TLink createdNumber = links.GetTarget(createdNumberValue);
269     TLink numberMarker = links.GetSource(createdNumber);
270     Assert.Equal(numberMarker, defaultJsonStorage.NumberMarker);
271
272     TLink foundNumberValue = defaultJsonStorage.GetValueLink(document);
273     Assert.Equal(createdNumberValue, foundNumberValue);
274 }
275
276 /// <summary>
277 /// <para>
278 /// Tests that attach true value to document test.
279 /// </para>
280 /// <para></para>
281 /// </summary>
282 [Fact]
283 public void AttachTrueValueToDocumentTest()
284 {
285     var links = CreateLinks();
286     var defaultJsonStorage = CreateJsonStorage(links);
287     TLink document = defaultJsonStorage.CreateDocument("documentName");
288
289     TLink documentTrueValueLink = defaultJsonStorage.AttachBoolean(document, true);
290     TLink createdTrueValue = links.GetTarget(documentTrueValueLink);

```

```

291     TLink valueMarker = links.GetSource(createdTrueValue);
292     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
293
294     TLink trueMarker = links.GetTarget(createdTrueValue);
295     Assert.Equal(trueMarker, defaultJsonStorage.TrueMarker);
296
297     TLink foundTrueValue = defaultJsonStorage.GetValueLink(document);
298     Assert.Equal(createdTrueValue, foundTrueValue);
299 }
300
301 /// <summary>
302 /// <para>
303 /// Tests that attach false value to document test.
304 /// </para>
305 /// <para></para>
306 /// </summary>
307 [Fact]
308 public void AttachFalseValueToDocumentTest()
309 {
310     var links = CreateLinks();
311     var defaultJsonStorage = CreateJsonStorage(links);
312     TLink document = defaultJsonStorage.CreateDocument("documentName");
313
314     TLink documentFalseValueLink = defaultJsonStorage.AttachBoolean(document, false);
315     TLink createdFalseValue = links.GetTarget(documentFalseValueLink);
316
317     TLink valueMarker = links.GetSource(createdFalseValue);
318     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
319
320     TLink falseMarker = links.GetTarget(createdFalseValue);
321     Assert.Equal(falseMarker, defaultJsonStorage.FalseMarker);
322
323     TLink foundFalseValue = defaultJsonStorage.GetValueLink(document);
324     Assert.Equal(createdFalseValue, foundFalseValue);
325 }
326
327 /// <summary>
328 /// <para>
329 /// Tests that attach null value to document test.
330 /// </para>
331 /// <para></para>
332 /// </summary>
333 [Fact]
334 public void AttachNullValueToDocumentTest()
335 {
336     var links = CreateLinks();
337     var defaultJsonStorage = CreateJsonStorage(links);
338     TLink document = defaultJsonStorage.CreateDocument("documentName");
339
340     TLink documentNullValueLink = defaultJsonStorage.AttachNull(document);
341     TLink createdNullValue = links.GetTarget(documentNullValueLink);
342
343     TLink valueMarker = links.GetSource(createdNullValue);
344     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
345
346     TLink nullMarker = links.GetTarget(createdNullValue);
347     Assert.Equal(nullMarker, defaultJsonStorage.NullMarker);
348
349     TLink foundNullValue = defaultJsonStorage.GetValueLink(document);
350     Assert.Equal(createdNullValue, foundNullValue);
351 }
352
353 /// <summary>
354 /// <para>
355 /// Tests that attach empty array value to document test.
356 /// </para>
357 /// <para></para>
358 /// </summary>
359 [Fact]
360 public void AttachEmptyArrayValueToDocumentTest()
361 {
362     var links = CreateLinks();
363     var defaultJsonStorage = CreateJsonStorage(links);
364     TLink document = defaultJsonStorage.CreateDocument("documentName");
365
366     TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, new
367         ↪ TLink[0]);
368     TLink createdArrayValue = links.GetTarget(documentArrayValueLink);

```

```

369     output.WriteLine(links.Format(createdArrayValue));
370
371
372     TLink valueMarker = links.GetSource(createdArrayValue);
373     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
374
375     TLink createdArrayLink = links.GetTarget(createdArrayValue);
376     TLink arrayMarker = links.GetSource(createdArrayLink);
377     Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
378
379     TLink createArrayContents = links.GetTarget(createdArrayLink);
380     Assert.Equal(createArrayContents, defaultJsonStorage.EmptyArrayMarker);
381
382     TLink foundArrayValue = defaultJsonStorage.GetValueLink(document);
383     Assert.Equal(createdArrayValue, foundArrayValue);
384 }
385
386 /// <summary>
387 /// <para>
388 /// Tests that attach array value to document test.
389 /// </para>
390 /// <para></para>
391 /// </summary>
392 [Fact]
393 public void AttachArrayValueToDocumentTest()
394 {
395     var links = CreateLinks();
396     var defaultJsonStorage = CreateJsonStorage(links);
397     TLink document = defaultJsonStorage.CreateDocument("documentName");
398
399     TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
400     TLink[] array = new TLink[] { arrayElement, arrayElement, arrayElement };
401
402
403     TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, array);
404     TLink createdArrayValue = links.GetTarget(documentArrayValueLink);
405
406     DefaultStack<TLink> stack = new();
407     RightSequenceWalker<TLink> rightSequenceWalker = new(links, stack, arrayElementLink
408         ↪ => links.GetSource(arrayElementLink) == defaultJsonStorage.ValueMarker);
409     IEnumerable<TLink> arrayElementsValuesLink =
410         ↪ rightSequenceWalker.Walk(createdArrayValue);
411     Assert.NotEmpty(arrayElementsValuesLink);
412
413     output.WriteLine(links.Format(createdArrayValue));
414
415
416     TLink valueMarker = links.GetSource(createdArrayValue);
417     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
418
419     TLink createdArrayLink = links.GetTarget(createdArrayValue);
420     TLink arrayMarker = links.GetSource(createdArrayLink);
421     Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
422
423     TLink createdArrayContents = links.GetTarget(createdArrayLink);
424     Assert.Equal(links.GetTarget(createdArrayContents), arrayElement);
425
426     TLink foundArrayValue = defaultJsonStorage.GetValueLink(document);
427     Assert.Equal(createdArrayValue, foundArrayValue);
428 }
429
430 /// <summary>
431 /// <para>
432 /// Tests that get object from document object value link test.
433 /// </para>
434 /// <para></para>
435 /// </summary>
436 [Fact]
437 public void GetObjectFromDocumentObjectValueLinkTest()
438 {
439     ILinks<TLink> links = CreateLinks();
440     var defaultJsonStorage = CreateJsonStorage(links);
441     TLink document = defaultJsonStorage.CreateDocument("documentName");
442     TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
443     TLink objectValueLink = links.GetTarget(documentObjectValueLink);
444     TLink objectFromGetObject = defaultJsonStorage.GetObject(documentObjectValueLink);
445     output.WriteLine(links.Format(objectValueLink));
446     output.WriteLine(links.Format(objectFromGetObject));

```



```

446     Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
447 }
448
449 /// <summary>
450 /// <para>
451 /// Tests that get object from object value link test.
452 /// </para>
453 /// <para></para>
454 /// </summary>
455 [Fact]
456 public void GetObjectFromObjectValueLinkTest()
457 {
458     ILinks<TLink> links = CreateLinks();
459     var defaultJsonStorage = CreateJsonStorage(links);
460     TLink document = defaultJsonStorage.CreateDocument("documentName");
461     TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
462     TLink objectValueLink = links.GetTarget(documentObjectValueLink);
463     TLink objectFromGetObject = defaultJsonStorage.GetObject(objectValueLink);
464     Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
465 }
466
467 /// <summary>
468 /// <para>
469 /// Tests that attach string value to key.
470 /// </para>
471 /// <para></para>
472 /// </summary>
473 [Fact]
474 public void AttachStringValueToKey()
475 {
476     ILinks<TLink> links = CreateLinks();
477     var defaultJsonStorage = CreateJsonStorage(links);
478     TLink document = defaultJsonStorage.CreateDocument("documentName");
479     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
480     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
481     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
482     TLink memberStringValueLink = defaultJsonStorage.AttachString(memberLink,
483         ↪ "stringValue");
484     TLink stringValueLink = links.GetTarget(memberStringValueLink);
485     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
486     Assert.Equal(memberLink, objectMembersLinks[0]);
487     Assert.Equal(stringValueLink,
488         ↪ defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
489 }
490
491 /// <summary>
492 /// <para>
493 /// Tests that attach number value to key.
494 /// </para>
495 /// <para></para>
496 /// </summary>
497 [Fact]
498 public void AttachNumberValueToKey()
499 {
500     ILinks<TLink> links = CreateLinks();
501     var defaultJsonStorage = CreateJsonStorage(links);
502     TLink document = defaultJsonStorage.CreateDocument("documentName");
503     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
504     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
505     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
506     TLink memberNumberValueLink = defaultJsonStorage.AttachNumber(memberLink, 123);
507     TLink numberValueLink = links.GetTarget(memberNumberValueLink);
508     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
509     Assert.Equal(memberLink, objectMembersLinks[0]);
510     Assert.Equal(numberValueLink,
511         ↪ defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
512 }
513
514 /// <summary>
515 /// <para>
516 /// Tests that attach object value to key.
517 /// </para>
518 /// <para></para>
519 /// </summary>
520 [Fact]
521 public void AttachObjectValueToKey()
522 {
523     ILinks<TLink> links = CreateLinks();

```

```

521     var defaultJsonStorage = CreateJsonStorage(links);
522     TLink document = defaultJsonStorage.CreateDocument("documentName");
523     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
524     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
525     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
526     TLink memberObjectValueLink = defaultJsonStorage.AttachObject(memberLink);
527     TLink objectValueLink = links.GetTarget(memberObjectValueLink);
528     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
529     Assert.Equal(memberLink, objectMembersLinks[0]);
530     Assert.Equal(objectValueLink,
    ↪     defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
531 }
532
533 /// <summary>
534 /// <para>
535 /// Tests that attach array value to key.
536 /// </para>
537 /// <para></para>
538 /// </summary>
539 [Fact]
540 public void AttachArrayValueToKey()
541 {
542     ILinks<TLink> links = CreateLinks();
543     var defaultJsonStorage = CreateJsonStorage(links);
544     TLink document = defaultJsonStorage.CreateDocument("documentName");
545     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
546     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
547     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
548     TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
549     TLink[] array = { arrayElement, arrayElement, arrayElement };
550     TLink memberArrayValueLink = defaultJsonStorage.AttachArray(memberLink, array);
551     TLink arrayValueLink = links.GetTarget(memberArrayValueLink);
552     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
553     Assert.Equal(memberLink, objectMembersLinks[0]);
554     Assert.Equal(arrayValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
555 }
556
557 /// <summary>
558 /// <para>
559 /// Tests that attach true value to key.
560 /// </para>
561 /// <para></para>
562 /// </summary>
563 [Fact]
564 public void AttachTrueValueToKey()
565 {
566     ILinks<TLink> links = CreateLinks();
567     var defaultJsonStorage = CreateJsonStorage(links);
568     TLink document = defaultJsonStorage.CreateDocument("documentName");
569     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
570     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
571     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
572     TLink memberTrueValueLink = defaultJsonStorage.AttachBoolean(memberLink, true);
573     TLink trueValueLink = links.GetTarget(memberTrueValueLink);
574     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
575     Assert.Equal(memberLink, objectMembersLinks[0]);
576     Assert.Equal(trueValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
577 }
578
579 /// <summary>
580 /// <para>
581 /// Tests that attach false value to key.
582 /// </para>
583 /// <para></para>
584 /// </summary>
585 [Fact]
586 public void AttachFalseValueToKey()
587 {
588     ILinks<TLink> links = CreateLinks();
589     var defaultJsonStorage = CreateJsonStorage(links);
590     TLink document = defaultJsonStorage.CreateDocument("documentName");
591     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
592     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
593     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
594     TLink memberFalseValueLink = defaultJsonStorage.AttachBoolean(memberLink, false);
595     TLink falseValueLink = links.GetTarget(memberFalseValueLink);
596     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
597     Assert.Equal(memberLink, objectMembersLinks[0]);

```

```

598     Assert.Equal(falseValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
599 }
600
601 /// <summary>
602 /// <para>
603 /// Tests that attach null value to key.
604 /// </para>
605 /// <para></para>
606 /// </summary>
607 [Fact]
608 public void AttachNullValueToKey()
609 {
610     ILinks<TLink> links = CreateLinks();
611     var defaultJsonStorage = CreateJsonStorage(links);
612     TLink document = defaultJsonStorage.CreateDocument("documentName");
613     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
614     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
615     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
616     TLink memberNullValueLink = defaultJsonStorage.AttachNull(memberLink);
617     TLink nullValueLink = links.GetTarget(memberNullValueLink);
618     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
619     Assert.Equal(nullValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
620 }
621 }
622 }

```

Index

./csharp/Platform.Data.Doublets.Json.Tests/JsonImportAndExportTests.cs, 32
./csharp/Platform.Data.Doublets.Json.Tests/JsonStorageTests.cs, 35
./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, 23
./csharp/Platform.Data.Doublets.Json/JsonExporterCli.cs, 27
./csharp/Platform.Data.Doublets.Json/JsonImporter.cs, 28
./csharp/Platform.Data.Doublets.Json/JsonImporterCli.cs, 31