

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

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

```
1 using Platform.Numbers;
2 using Platform.Data.Doublents.Unicode;
3 using Platform.Data.Doublents.Sequences.Converters;
4 using Platform.Data.Doublents.CriterionMatchers;
5 using Platform.Data.Numbers.Raw;
6 using Platform.Converters;
7 using Platform.Data.Doublents.Sequences.Walkers;
8 using Platform.Collections.Stacks;
9 using System;
10 using System.Collections.Generic;
11 using Platform.Data.Doublents.Numbers.Rational;
12 using Platform.Data.Doublents.Numbers.Raw;
13 using Platform.Data.Doublents.Sequences.HeightProviders;
14 using Platform.Data.Doublents.Sequences;
15
16 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
17
18 namespace Platform.Data.Doublents.Json
19 {
20     public class DefaultJsonStorage<TLink> : IJsonStorage<TLink>
21         where TLink : struct
22     {
23         public readonly TLink Any;
24         public static readonly TLink Zero = default;
25         public static readonly TLink One = Arithmetic.Increment(Zero);
26         public readonly BalancedVariantConverter<TLink> BalancedVariantConverter;
27         public readonly IConverter<IList<TLink>, TLink> ListToSequenceConverter;
28         public readonly TLink MeaningRoot;
29         public readonly EqualityComparer<TLink> EqualityComparer =
30             ↳ EqualityComparer<TLink>.Default;
31         // Converters that are able to convert link's address (UInt64 value) to a raw number
32         ↳ represented with another UInt64 value and back
33         public readonly RawNumberToAddressConverter<TLink> NumberToAddressConverter = new();
34         public readonly AddressToRawNumberConverter<TLink> AddressToNumberConverter = new();
35         // Converters between BigInteger and raw number sequence
36         public readonly BigIntegerToRawNumberSequenceConverter<TLink>
37             ↳ BigIntegerToRawNumberSequenceConverter;
38         public readonly RawNumberSequenceToBigIntegerConverter<TLink>
39             ↳ RawNumberSequenceToBigIntegerConverter;
40         // Converters between decimal and rational number sequence
41         public readonly DecimalToRationalConverter<TLink> DecimalToRationalConverter;
42         public readonly RationalToDecimalConverter<TLink> RationalToDecimalConverter;
43         // Converters between string and unicode sequence
44         public readonly IConverter<string, TLink> StringToUnicodeSequenceConverter;
45         public readonly IConverter<TLink, string> UnicodeSequenceToStringConverter;
46         // For sequences
47         public readonly JsonArrayElementCriterionMatcher<TLink> JsonArrayElementCriterionMatcher;
48         public readonly DefaultSequenceRightHeightProvider<TLink>
49             ↳ DefaultSequenceRightHeightProvider;
50         public readonly DefaultSequenceAppender<TLink> DefaultSequenceAppender;
51         public IList<TLink> Links { get; }
52         public TLink DocumentMarker { get; }
53         public TLink ObjectMarker { get; }
54         public TLink MemberMarker { get; }
55         public TLink ValueMarker { get; }
56         public TLink StringMarker { get; }
57         public TLink EmptyStringMarker { get; }
58         public TLink NumberMarker { get; }
59         public TLink NegativeNumberMarker { get; }
60         public TLink ArrayMarker { get; }
61         public TLink EmptyArrayMarker { get; }
62         public TLink TrueMarker { get; }
63         public TLink FalseMarker { get; }
64         public TLink NullMarker { get; }
65
66         public DefaultJsonStorage(IList<TLink> links, IConverter<IList<TLink>, TLink>
67             ↳ listToSequenceConverter)
68         {
69             Links = links;
70             ListToSequenceConverter = listToSequenceConverter;
71             // Initializes constants
72             Any = Links.Constants.Any;
73             var markerIndex = One;
74             MeaningRoot = links.GetOrCreate(markerIndex, markerIndex);
75             var unicodeSymbolMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
76                 ↳ markerIndex));
77             var unicodeSequenceMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
78                 ↳ markerIndex));
79         }
80     }
81 }
```

```

71     DocumentMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
    ↪ markerIndex));
72     ObjectMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
73     MemberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
74     ValueMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
75     StringMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
76     EmptyStringMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
    ↪ markerIndex));
77     NumberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
78     NegativeNumberMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
    ↪ markerIndex));
79     ArrayMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
80     EmptyArrayMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
    ↪ markerIndex));
81     TrueMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
82     FalseMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
83     NullMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref markerIndex));
84     BalancedVariantConverter = new(links);
85     TargetMatcher<TLink> unicodeSymbolCriterionMatcher = new(Links, unicodeSymbolMarker);
86     TargetMatcher<TLink> unicodeSequenceCriterionMatcher = new(Links,
    ↪ unicodeSequenceMarker);
87     CharToUnicodeSymbolConverter<TLink> charToUnicodeSymbolConverter =
88         new(Links, AddressToNumberConverter, unicodeSymbolMarker);
89     UnicodeSymbolToCharConverter<TLink> unicodeSymbolToCharConverter =
90         new(Links, NumberToAddressConverter, unicodeSymbolCriterionMatcher);
91     StringToUnicodeSequenceConverter = new CachingConverterDecorator<string, TLink>(
92         new StringToUnicodeSequenceConverter<TLink>(Links, charToUnicodeSymbolConverter,
93             BalancedVariantConverter, unicodeSequenceMarker));
94     RightSequenceWalker<TLink> sequenceWalker =
95         new(Links, new DefaultStack<TLink>(), unicodeSymbolCriterionMatcher.IsMatched);
96     UnicodeSequenceToStringConverter = new CachingConverterDecorator<TLink, string>(
97         new UnicodeSequenceToStringConverter<TLink>(Links,
    ↪ unicodeSequenceCriterionMatcher, sequenceWalker,
98             unicodeSymbolToCharConverter));
99     BigIntegerToRawNumberSequenceConverter =
100         new(links, AddressToNumberConverter, ListToSequenceConverter,
    ↪ NegativeNumberMarker);
101     RawNumberSequenceToBigIntegerConverter = new(links, NumberToAddressConverter,
    ↪ NegativeNumberMarker);
102     DecimalToRationalConverter = new(links, BigIntegerToRawNumberSequenceConverter);
103     RationalToDecimalConverter = new(links, RawNumberSequenceToBigIntegerConverter);
104     JsonArrayElementCriterionMatcher = new(this);
105     DefaultSequenceRightHeightProvider = new(Links, JsonArrayElementCriterionMatcher);
106     DefaultSequenceAppender = new(Links, new DefaultStack<TLink>(),
    ↪ DefaultSequenceRightHeightProvider);
107 }
108
109 public TLink CreateString(string content)
110 {
111     var @string = GetStringSequence(content);
112     return Links.GetOrCreate(StringMarker, @string);
113 }
114
115 public TLink CreateStringValue(string content)
116 {
117     var @string = CreateString(content);
118     return CreateValue(@string);
119 }
120
121 public TLink CreateNumber(decimal number)
122 {
123     var numberSequence = DecimalToRationalConverter.Convert(number);
124     return Links.GetOrCreate(NumberMarker, numberSequence);
125 }
126
127 public TLink CreateNumberValue(decimal number)
128 {
129     var numberLink = CreateNumber(number);
130     return CreateValue(numberLink);
131 }
132
133 public TLink CreateBooleanValue(bool value) => CreateValue(value ? TrueMarker :
    ↪ FalseMarker);
134
135 public TLink CreateNullValue() => CreateValue(NullMarker);
136
137 public TLink CreateDocument(string name)
138 {

```

```

139     var documentName = CreateString(name);
140     return Links.GetOrCreate(DocumentMarker, documentName);
141 }
142
143 public TLink CreateObject()
144 {
145     var @object = Links.Create();
146     return Links.Update(@object, newSource: ObjectMarker, newTarget: @object);
147 }
148
149 public TLink CreateObjectValue()
150 {
151     var @object = CreateObject();
152     return CreateValue(@object);
153 }
154
155 public TLink CreateArray(ICollection<TLink> array)
156 {
157     var arraySequence = array.Count == 0 ? EmptyArrayMarker :
        ↳ BalancedVariantConverter.Convert(array);
158     return CreateArray(arraySequence);
159 }
160
161 public TLink CreateArray(TLink sequence) => Links.GetOrCreate(ArrayMarker, sequence);
162
163 public TLink CreateArrayValue(ICollection<TLink> array)
164 {
165     var arrayLink = CreateArray(array);
166     return CreateValue(arrayLink);
167 }
168
169 public TLink CreateArrayValue(TLink sequence)
170 {
171     var array = CreateArray(sequence);
172     return CreateValue(array);
173 }
174
175 public TLink CreateMember(string name)
176 {
177     var nameLink = CreateString(name);
178     return Links.GetOrCreate(MemberMarker, nameLink);
179 }
180
181 public TLink CreateValue(TLink value) => Links.GetOrCreate(ValueMarker, value);
182
183 public TLink AttachObject(TLink parent) => Attach(parent, CreateObjectValue());
184
185 public TLink AttachString(TLink parent, string content)
186 {
187     var @string = CreateString(content);
188     var stringValue = CreateValue(@string);
189     return Attach(parent, stringValue);
190 }
191
192 public TLink AttachNumber(TLink parent, decimal number)
193 {
194     var numberLink = CreateNumber(number);
195     var numberValue = CreateValue(numberLink);
196     return Attach(parent, numberValue);
197 }
198
199 public TLink AttachBoolean(TLink parent, bool value)
200 {
201     var booleanValue = CreateBooleanValue(value);
202     return Attach(parent, booleanValue);
203 }
204
205 public TLink AttachNull(TLink parent)
206 {
207     var nullValue = CreateNullValue();
208     return Attach(parent, nullValue);
209 }
210
211 public TLink AttachArray(TLink parent, ICollection<TLink> array)
212 {
213     var arrayValue = CreateArrayValue(array);
214     return Attach(parent, arrayValue);
215 }
216

```

```

217 public TLink AttachMemberToObject(TLink @object, string keyName)
218 {
219     var member = CreateMember(keyName);
220     return Attach(@object, member);
221 }
222
223 public TLink Attach(TLink parent, TLink child) => Links.GetOrCreate(parent, child);
224
225 public TLink AppendArrayValue(TLink arrayValue, TLink appendant)
226 {
227     var array = GetArray(arrayValue);
228     var arraySequence = Links.GetTarget(array);
229     TLink newArrayValue;
230     if (EqualityComparer.Equals(arraySequence, EmptyArrayMarker))
231     {
232         newArrayValue = CreateArrayValue(appendant);
233     }
234     else
235     {
236         arraySequence = DefaultSequenceAppender.Append(arraySequence, appendant);
237         newArrayValue = CreateArrayValue(arraySequence);
238     }
239     return newArrayValue;
240 }
241
242 public TLink GetDocumentOrDefault(string name)
243 {
244     var stringSequence = GetStringSequence(name);
245     var @string = Links.SearchOrDefault(StringMarker, stringSequence);
246     if (EqualityComparer.Equals(@string, default))
247     {
248         return default;
249     }
250     return Links.SearchOrDefault(DocumentMarker, @string);
251 }
252
253 private TLink GetStringSequence(string content) => content == "" ? EmptyStringMarker :
    ↳ StringToUnicodeSequenceConverter.Convert(content);
254
255 public string GetString(TLink stringValue)
256 {
257     var current = stringValue;
258     TLink source;
259     for (int i = 0; i < 3; i++)
260     {
261         source = Links.GetSource(current);
262         if (EqualityComparer.Equals(source, StringMarker))
263         {
264             var sequence = Links.GetTarget(current);
265             var isEmpty = EqualityComparer.Equals(sequence, EmptyStringMarker);
266             return isEmpty ? "" : UnicodeSequenceToStringConverter.Convert(sequence);
267         }
268         current = Links.GetTarget(current);
269     }
270     throw new Exception("The passed link does not contain a string.");
271 }
272
273 public decimal GetNumber(TLink valueLink)
274 {
275     var current = valueLink;
276     TLink source;
277     TLink target;
278     for (int i = 0; i < 3; i++)
279     {
280         source = Links.GetSource(current);
281         target = Links.GetTarget(current);
282         if (EqualityComparer.Equals(source, NumberMarker))
283         {
284             return RationalToDecimalConverter.Convert(target);
285         }
286         current = target;
287     }
288     throw new Exception("The passed link does not contain a number.");
289 }
290
291 public TLink GetObject(TLink objectValueLink)
292 {
293     var current = objectValueLink;
294

```

```

295     TLink source;
296     for (int i = 0; i < 3; i++)
297     {
298         source = Links.GetSource(current);
299         if (EqualityComparer.Equals(source, ObjectMarker))
300         {
301             return current;
302         }
303         current = Links.GetTarget(current);
304     }
305     throw new Exception("The passed link does not contain an object.");
306 }
307
308 public TLink GetArray(TLink arrayValueLink)
309 {
310     var current = arrayValueLink;
311     TLink source;
312     for (int i = 0; i < 3; i++)
313     {
314         source = Links.GetSource(current);
315         if (EqualityComparer.Equals(source, ArrayMarker))
316         {
317             return current;
318         }
319         current = Links.GetTarget(current);
320     }
321     throw new Exception("The passed link does not contain an array.");
322 }
323
324 public TLink GetArraySequence(TLink array) => Links.GetTarget(array);
325
326 public TLink GetValueLink(TLink parent)
327 {
328     var query = new Link<TLink>(index: Any, source: parent, target: Any);
329     var resultLinks = Links.All(query);
330     switch (resultLinks.Count)
331     {
332         case 0:
333             return default;
334         case 1:
335             var resultLinkTarget = Links.GetTarget(resultLinks[0]);
336             if (EqualityComparer.Equals(Links.GetSource(resultLinkTarget), ValueMarker))
337             {
338                 return resultLinkTarget;
339             }
340             else
341             {
342                 throw new InvalidOperationException("The passed link is not a value.");
343             }
344         case > 1:
345             throw new InvalidOperationException("More than 1 value found.");
346         default:
347             throw new InvalidOperationException("The list elements length is negative.");
348     }
349 }
350
351 public TLink GetValueMarker(TLink value)
352 {
353     var target = Links.GetTarget(value);
354     var targetSource = Links.GetSource(target);
355     if (EqualityComparer.Equals(MeaningRoot, targetSource))
356     {
357         return target;
358     }
359     return targetSource;
360 }
361
362 public List<TLink> GetMembersLinks(TLink @object)
363 {
364     Link<TLink> query = new(index: Any, source: @object, target: Any);
365     List<TLink> members = new();
366     Links.Each(objectMemberLink =>
367     {
368         var memberLink = Links.GetTarget(objectMemberLink);
369         var memberMarker = Links.GetSource(memberLink);
370         if (EqualityComparer.Equals(memberMarker, MemberMarker))
371         {
372             members.Add(Links.GetIndex(objectMemberLink));
373         }
374     });

```

```

374         return Links.Constants.Continue;
375     }, query);
376     return members;
377 }
378 }
379 }

```

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     public interface IJsonStorage<TLink>
8     {
9         public ILinks<TLink> Links { get; }
10        public TLink DocumentMarker { get; }
11        public TLink ObjectMarker { get; }
12        public TLink StringMarker { get; }
13        public TLink EmptyStringMarker { get; }
14        public TLink MemberMarker { get; }
15        public TLink ValueMarker { get; }
16        public TLink NumberMarker { get; }
17        public TLink ArrayMarker { get; }
18        public TLink EmptyArrayMarker { get; }
19        public TLink TrueMarker { get; }
20        public TLink FalseMarker { get; }
21        public TLink NullMarker { get; }
22        TLink CreateString(string content);
23        TLink CreateStringValue(string content);
24        TLink CreateNumber(decimal number);
25        TLink CreateNumberValue(decimal number);
26        TLink CreateBooleanValue(bool value);
27        TLink CreateNullValue();
28        TLink CreateDocument(string name);
29        TLink GetDocumentOrDefault(string name);
30        TLink CreateObject();
31        TLink CreateObjectValue();
32        TLink CreateArray(IList<TLink> array);
33        TLink CreateArrayValue(IList<TLink> array) => CreateValue(CreateArray(array));
34        TLink CreateArrayValue(TLink array) => CreateValue(array);
35        TLink CreateMember(string name);
36        TLink CreateValue(TLink value);
37        TLink Attach(TLink source, TLink target);
38        TLink AttachObject(TLink parent);
39        TLink AttachString(TLink parent, string content);
40        TLink AttachNumber(TLink parent, decimal number);
41        TLink AttachBoolean(TLink parent, bool value);
42        TLink AttachNull(TLink parent);
43        TLink AttachArray(TLink parent, IList<TLink> array);
44        TLink AttachMemberToObject(TLink @object, string keyName);
45        TLink AppendArrayValue(TLink arrayValue, TLink appendant);
46        string GetString(TLink stringValue);
47        decimal GetNumber(TLink value);
48        TLink GetObject(TLink objectValue);
49        TLink GetArray(TLink arrayValueLink);
50        TLink GetArraySequence(TLink array);
51        TLink GetValueLink(TLink parent);
52        TLink GetValueMarker(TLink link);
53        List<TLink> GetMembersLinks(TLink @object);
54    }
55 }

```

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     public class JsonArrayElementCriterionMatcher<TLink> : ICriterionMatcher<TLink>
21     {
22         public readonly IJsonStorage<TLink> Storage;
23         public JsonArrayElementCriterionMatcher(IJsonStorage<TLink> storage) => Storage =
            ↳ storage;
24         public bool IsMatched(TLink link) =>
            ↳ EqualityComparer<TLink>.Default.Equals(Storage.Links.GetSource(link),
            ↳ Storage.ValueMarker);
25     }
26 }

```

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     public class JsonExporter<TLink>
13     {
14         public readonly IJsonStorage<TLink> Storage;
15         public readonly EqualityComparer<TLink> EqualityComparer =
            ↳ EqualityComparer<TLink>.Default;
16
17         public JsonExporter(IJsonStorage<TLink> storage) => Storage = storage;
18
19         private bool IsElement(TLink link)
20         {
21             var marker = Storage.Links.GetSource(link);
22             return EqualityComparer.Equals(marker, Storage.ValueMarker);
23         }
24
25         private void WriteStringValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
            ↳ utf8JsonWriter.WriteStringValue(Storage.GetString(valueLink));
26
27         private void WriteString(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
            ↳ valueLink) => utf8JsonWriter.WriteString(parent, Storage.GetString(valueLink));
28
29         private void WriteNumberValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
            ↳ utf8JsonWriter.WriteNumberValue(Storage.GetNumber(valueLink));
30
31         private void WriteNumber(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
            ↳ valueLink) => utf8JsonWriter.WriteNumber(parent, Storage.GetNumber(valueLink));
32
33         private void Write(ref Utf8JsonWriter utf8JsonWriter, string parent, TLink valueLink,
            ↳ CancellationToken cancellationToken)
34         {
35             if (cancellationToken.IsCancellationRequested)
36             {
37                 return;
38             }
39             var valueMarker = Storage.GetValueMarker(valueLink);
40             if (EqualityComparer.Equals(valueMarker, Storage.ObjectMarker))
41             {
42                 utf8JsonWriter.WriteStartObject(parent);
43                 var membersLinks = Storage.GetMembersLinks(Storage.GetObject(valueLink));
44                 foreach (var memberLink in membersLinks)
45                 {
46                     if (cancellationToken.IsCancellationRequested)
47                     {
48                         return;
49                     }
50                     Write(ref utf8JsonWriter, Storage.GetString(memberLink),
                        ↳ Storage.GetValueLink(memberLink), cancellationToken);
51                 }
52                 utf8JsonWriter.WriteEndObject();
53             }
54             else if (EqualityComparer.Equals(valueMarker, Storage.ArrayMarker))
55             {

```

```

56     var array = Storage.GetArray(valueLink);
57     var sequence = Storage.GetArraySequence(array);
58     utf8JsonWriter.WriteStartArray(parent);
59     if (!EqualityComparer.Equals(sequence, Storage.EmptyArrayMarker))
60     {
61         RightSequenceWalker<TLink> rightSequenceWalker = new(Storage.Links, new
        ↪     DefaultStack<TLink>(), IsElement);
62         var elements = rightSequenceWalker.Walk(sequence);
63         foreach (var element in elements)
64         {
65             if (cancellationToken.IsCancellationRequested)
66             {
67                 return;
68             }
69             Write(ref utf8JsonWriter, element, in cancellationToken);
70         }
71     }
72     utf8JsonWriter.WriteEndArray();
73 }
74 else if (EqualityComparer.Equals(valueMarker, Storage.StringMarker))
75 {
76     WriteString(in utf8JsonWriter, parent, valueLink);
77 }
78 else if (EqualityComparer.Equals(valueMarker, Storage.NumberMarker))
79 {
80     WriteNumber(in utf8JsonWriter, parent, valueLink);
81 }
82 else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
83 {
84     utf8JsonWriter.WriteBoolean(parent, true);
85 }
86 else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
87 {
88     utf8JsonWriter.WriteBoolean(parent, false);
89 }
90 else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
91 {
92     utf8JsonWriter.WriteNull(parent);
93 }
94 }
95
96 private void Write(ref Utf8JsonWriter utf8JsonWriter, TLink valueLink, in
    ↪ Cancellation token cancellationToken)
97 {
98     if (cancellationToken.IsCancellationRequested)
99     {
100         return;
101     }
102     var valueMarker = Storage.GetValueMarker(valueLink);
103     if (EqualityComparer.Equals(valueMarker, Storage.ObjectMarker))
104     {
105         utf8JsonWriter.WriteStartObject();
106         var membersLinks = Storage.GetMembersLinks(Storage.GetObject(valueLink));
107         foreach (var memberLink in membersLinks)
108         {
109             if (cancellationToken.IsCancellationRequested)
110             {
111                 return;
112             }
113             Write(ref utf8JsonWriter, Storage.GetString(memberLink),
                ↪ Storage.GetValueLink(memberLink), cancellationToken);
114         }
115         utf8JsonWriter.WriteEndObject();
116     }
117     else if (EqualityComparer.Equals(valueMarker, Storage.ArrayMarker))
118     {
119         var array = Storage.GetArray(valueLink);
120         var sequence = Storage.GetArraySequence(array);
121         utf8JsonWriter.WriteStartArray();
122         if (!EqualityComparer.Equals(sequence, Storage.EmptyArrayMarker))
123         {
124             RightSequenceWalker<TLink> rightSequenceWalker = new(Storage.Links, new
                ↪     DefaultStack<TLink>(), IsElement);
125             var elements = rightSequenceWalker.Walk(sequence);
126             foreach (var element in elements)
127             {
128                 if (cancellationToken.IsCancellationRequested)
129                 {

```



```

130         return;
131     }
132     Write(ref utf8JsonWriter, element, in cancellationToken);
133 }
134 }
135 utf8JsonWriter.WriteEndArray();
136 }
137 else if (EqualityComparer.Equals(valueMarker, Storage.StringMarker))
138 {
139     WriteStringValue(in utf8JsonWriter, valueLink);
140 }
141 else if (EqualityComparer.Equals(valueMarker, Storage.NumberMarker))
142 {
143     WriteNumberValue(in utf8JsonWriter, valueLink);
144 }
145 else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
146 {
147     utf8JsonWriter.WriteBooleanValue(true);
148 }
149 else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
150 {
151     utf8JsonWriter.WriteBooleanValue(false);
152 }
153 else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
154 {
155     utf8JsonWriter.WriteNullValue();
156 }
157 }
158
159 public void Export(TLink document, ref Utf8JsonWriter utf8JsonWriter, in
    ↳ Cancellation token cancellationToken)
160 {
161     if (EqualityComparer.Equals(document, default))
162     {
163         throw new Exception("No document with this name exists");
164     }
165     var valueLink = Storage.GetValueLink(document);
166     Write(ref utf8JsonWriter, valueLink, in cancellationToken);
167     utf8JsonWriter.Flush();
168 }
169
170 public void Export(string documentName, Utf8JsonWriter utf8JsonWriter, Cancellation token
    ↳ cancellationToken) => Export(Storage.GetDocumentOrDefault(documentName), ref
    ↳ utf8JsonWriter, in cancellationToken);
171 }
172 }

```

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     public class JsonExporterCli<TLink>
16         where TLink : struct
17     {
18         public void Run(params string[] args)
19         {
20             var argumentIndex = 0;
21             var linksFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Links file
    ↳ path", args);
22             var jsonFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "JSON file
    ↳ path", args);
23             var defaultDocumentName = Path.GetFileNameWithoutExtension(jsonFilePath);
24             var documentName = ConsoleHelpers.GetOrReadArgument(argumentIndex, $"Document name
    ↳ (default: {defaultDocumentName})", args);
25             if (string.IsNullOrEmpty(documentName))
26             {
27                 documentName = defaultDocumentName;
28             }

```

```

29         if (!File.Exists(linksFilePath))
30         {
31             Console.WriteLine($"${linksFilePath} file does not exist.");
32         }
33         using FileStream jsonFileStream = new(jsonFilePath, FileMode.Append);
34         JsonSerializerOptions utf8JsonWriterOptions = new()
35         {
36             Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping,
37             Indented = true
38         };
39         Utf8JsonWriter utf8JsonWriter = new(jsonFileStream, utf8JsonWriterOptions);
40         var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
41             ↪ true);
42         using UnitedMemoryLinks<TLink> memoryAdapter = new (new
43             ↪ FileMappedResizableDirectMemory(linksFilePath),
44             ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
45             ↪ IndexTreeType.Default);
46         var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
47         BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
48         var storage = new DefaultJsonStorage<TLink>(links, balancedVariantConverter);
49         var exporter = new JsonExporter<TLink>(storage);
50         var document = storage.GetDocumentOrDefault(documentName);
51         if (storage.EqualityComparer.Equals(document, default))
52         {
53             Console.WriteLine("No document with this name.");
54         }
55         using ConsoleCancellation cancellation = new ();
56         var cancellationToken = cancellation.Token;
57         Console.WriteLine("Press CTRL+C to stop.");
58         try
59         {
60             exporter.Export(document, ref utf8JsonWriter, in cancellationToken);
61         }
62         catch (Exception exception)
63         {
64             Console.WriteLine(exception);
65             return;
66         }
67         finally
68         {
69             utf8JsonWriter.Dispose();
70         }
71     }
72 }

```

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     public class JsonImporter<TLink>
11     {
12         public readonly IJsonStorage<TLink> Storage;
13         public readonly EqualityComparer<TLink> EqualityComparer =
14             ↪ EqualityComparer<TLink>.Default;
15         public readonly Stack<TLink> Parents = new ();
16         public JsonImporter(IJsonStorage<TLink> storage) => Storage = storage;
17
18         private void PopIfParentIsMember()
19         {
20             var parent = Parents.Peek();
21             var parentMarker = Storage.GetValueMarker(parent);
22             if (EqualityComparer.Equals(parentMarker, Storage.MemberMarker))
23             {
24                 Parents.Pop();
25             }
26         }
27
28         public TLink Import(string documentName, ref Utf8JsonReader utf8JsonReader, in
29             ↪ Cancellation token cancellationToken)
30         {
31             Parents.Clear();
32         }
33     }
34 }

```

```

30 if (!EqualityComparer.Equals(Storage.GetDocumentOrDefault(documentName), default))
31 {
32     throw new Exception("The document with the specified name already exists.");
33 }
34 var document = Storage.CreateDocument(documentName);
35 Parents.Push(document);
36 TLink parent;
37 TLink parentMarker;
38 JsonTokenType tokenType;
39 TLink value;
40 TLink newParentArray;
41 while (utf8JsonReader.Read())
42 {
43     cancellationToken.ThrowIfCancellationRequested();
44     parent = Parents.Peek();
45     parentMarker = Storage.GetValueMarker(parent);
46     tokenType = utf8JsonReader.TokenType;
47     if (utf8JsonReader.TokenType == JsonTokenType.PropertyName)
48     {
49         var @object = Storage.GetObject(parent);
50         var property = utf8JsonReader.GetString();
51         Parents.Push(Storage.AttachMemberToObject(@object, property));
52     }
53     switch (tokenType)
54     {
55     case JsonTokenType.StartObject:
56     {
57         value = Storage.CreateObjectValue();
58         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
59         {
60             Parents.Pop();
61             newParentArray = Storage.AppendArrayValue(parent, value);
62             Parents.Push(newParentArray);
63             Parents.Push(value);
64         }
65         else
66         {
67             var @object = Storage.Attach(parent, value);
68             Parents.Push(@object);
69         }
70         break;
71     }
72     case JsonTokenType.EndObject:
73         Parents.Pop();
74         break;
75     case JsonTokenType.StartArray:
76         value = Storage.CreateArrayValue(Array.Empty<TLink>());
77         Parents.Push(value);
78         break;
79     case JsonTokenType.EndArray:
80     {
81         var arrayValue = Parents.Pop();
82         parent = Parents.Peek();
83         parentMarker = Storage.GetValueMarker(parent);
84         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
85         {
86             Parents.Pop();
87             newParentArray = Storage.AppendArrayValue(parent, arrayValue);
88             Parents.Push(newParentArray);
89         }
90         Storage.Attach(parent, arrayValue);
91         break;
92     }
93     case JsonTokenType.String:
94     {
95         var @string = utf8JsonReader.GetString();
96         value = Storage.CreateStringValue(@string);
97         if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
98         {
99             Parents.Pop();
100             newParentArray = Storage.AppendArrayValue(parent, value);
101             Parents.Push(newParentArray);
102         }
103         else
104         {
105             Storage.Attach(parent, value);
106         }
107         break;
108     }
109 }

```

```

109         case JsonTokenType.Number:
110         {
111             value = Storage.CreateNumberValue(utf8JsonReader.GetDecimal());
112             if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
113             {
114                 Parents.Pop();
115                 newParentArray = Storage.AppendArrayValue(parent, value);
116                 Parents.Push(newParentArray);
117             }
118             else
119             {
120                 Storage.Attach(parent, value);
121             }
122             break;
123         }
124         case JsonTokenType.True:
125         {
126             value = Storage.CreateBooleanValue(true);
127             if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
128             {
129                 Parents.Pop();
130                 newParentArray = Storage.AppendArrayValue(parent, value);
131                 Parents.Push(newParentArray);
132             }
133             else
134             {
135                 Storage.Attach(parent, value);
136             }
137             break;
138         }
139         case JsonTokenType.False:
140         {
141             value = Storage.CreateBooleanValue(false);
142             if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
143             {
144                 Parents.Pop();
145                 newParentArray = Storage.AppendArrayValue(parent, value);
146                 Parents.Push(newParentArray);
147             }
148             else
149             {
150                 Storage.Attach(parent, value);
151             }
152             break;
153         }
154         case JsonTokenType.Null:
155         {
156             value = Storage.CreateNullValue();
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     }
170     if (tokenType != JsonTokenType.PropertyName && tokenType !=
171         ↪ JsonTokenType.StartObject && tokenType != JsonTokenType.StartArray)
172     {
173         PopIfParentIsMember();
174     }
175     return document;
176 }
177 }
178 }

```

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     public class JsonImporterCli<TLink>
16         where TLink : struct
17     {
18         public void Run(params string[] args)
19         {
20             var argumentIndex = 0;
21             var jsonFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "JSON file
22             ↪ path", args);
23             var linksFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Links file
24             ↪ path", args);
25             var defaultDocumentName = Path.GetFileNameWithoutExtension(jsonFilePath);
26             var documentName = ConsoleHelpers.GetOrReadArgument(argumentIndex, $"Document name
27             ↪ (default: {defaultDocumentName})", args);
28             if (string.IsNullOrEmpty(documentName))
29             {
30                 documentName = defaultDocumentName;
31             }
32             if (!File.Exists(jsonFilePath))
33             {
34                 Console.WriteLine($"${jsonFilePath} file does not exist.");
35             }
36             var json = File.ReadAllText(jsonFilePath);
37             var encodedJson = Encoding.UTF8.GetBytes(json);
38             ReadOnlySpan<byte> readOnlySpanEncodedJson = new(encodedJson);
39             Utf8JsonReader utf8JsonReader = new(readOnlySpanEncodedJson);
40             LinksConstants<TLink> linksConstants = new(enableExternalReferencesSupport: true);
41             FileMappedResizableDirectMemory fileMappedResizableDirectMemory = new(linksFilePath);
42             var unitedMemoryLinks = UnitedMemoryLinks<TLink>.DefaultLinksSizeStep;
43             const IndexTreeType indexTreeType = IndexTreeType.Default;
44             using UnitedMemoryLinks<TLink> memoryAdapter = new(fileMappedResizableDirectMemory,
45             ↪ unitedMemoryLinks, linksConstants, indexTreeType);
46             var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
47             BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
48             DefaultJsonStorage<TLink> storage = new(links, balancedVariantConverter);
49             JsonImporter<TLink> importer = new(storage);
50             using ConsoleCancellation cancellation = new();
51             var cancellationToken = cancellation.Token;
52             Console.WriteLine("Press CTRL+C to stop.");
53             try
54             {
55                 importer.Import(documentName, ref utf8JsonReader, in cancellationToken);
56             }
57             catch (Exception exception)
58             {
59                 Console.WriteLine(exception);
60                 return;
61             }
62             Console.WriteLine("Import completed successfully.");
63         }
64     }
65 }

```

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     public class JsonImportAndExportTests
16     {
17         public static BalancedVariantConverter<TLink> BalancedVariantConverter;
18
19         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new IO.TemporaryFile());
20     }
21 }

```

```

20
21 public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
22 {
23     var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
24         ↪ true);
25     return new UnitedMemoryLinks<TLink>(new
26         ↪ FileMappedResizableDirectMemory(dataDBFilename),
27         ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
28         ↪ IndexTreeType.Default);
29 }
30
31 public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links) => new
32     ↪ (links, BalancedVariantConverter);
33
34 public TLink Import(IJsonStorage<TLink> storage, string documentName, byte[] json)
35 {
36     Utf8JsonReader utf8JsonReader = new(json);
37     JsonImporter<TLink> jsonImporter = new(storage);
38     CancellationTokenSource importCancellationTokenSource = new();
39     CancellationToken cancellationToken = importCancellationTokenSource.Token;
40     return jsonImporter.Import(documentName, ref utf8JsonReader, in cancellationToken);
41 }
42
43 public void Export(TLink documentLink, IJsonStorage<TLink> storage, in MemoryStream
44     ↪ stream)
45 {
46     Utf8JsonWriter writer = new(stream);
47     JsonExporter<TLink> jsonExporter = new(storage);
48     CancellationTokenSource exportCancellationTokenSource = new();
49     CancellationToken exportCancellationToken = exportCancellationTokenSource.Token;
50     jsonExporter.Export(documentLink, ref writer, in exportCancellationToken);
51     writer.Dispose();
52 }
53
54 [Theory]
55 [InlineData("{}")]
56 [InlineData("{\"stringValue\":\"\"}")]
57 [InlineData("228")]
58 [InlineData("0.5")]
59 [InlineData("[ ]")]
60 [InlineData("true")]
61 [InlineData("false")]
62 [InlineData("null")]
63 [InlineData("{ \"string\": \"string\" }")]
64 [InlineData("{ \"null\": null }")]
65 [InlineData("{ \"boolean\": false }")]
66 [InlineData("{ \"boolean\": true }")]
67 [InlineData("{ \"array\": [ ] }")]
68 [InlineData("{ \"array\": [1] }")]
69 [InlineData("{ \"object\": { } }")]
70 [InlineData("{ \"number\": 1 }")]
71 [InlineData("{ \"decimal\": 0.5 }")]
72 [InlineData("[null]")]
73 [InlineData("[true]")]
74 [InlineData("[false]")]
75 [InlineData("[[]]")]
76 [InlineData("[[1]]")]
77 [InlineData("[[0.5]]")]
78 [InlineData("[{}]")]
79 [InlineData("[\"The Venus Project\"]")]
80 [InlineData("[{ \"title\": \"The Venus Project\" } ]")]
81 [InlineData("[1,2,3,4]")]
82 [InlineData("[-0.5, 0.5]")]
83 public void Test(string initialJson)
84 {
85     var links = CreateLinks();
86     BalancedVariantConverter = new(links);
87     var storage = CreateJsonStorage(links);
88     var json = Encoding.UTF8.GetBytes(initialJson);
89     var documentLink = Import(storage, "documentName", json);
90     MemoryStream stream = new();
91     Export(documentLink, storage, in stream);
92     string exportedJson = Encoding.UTF8.GetString(stream.ToArray());
93     stream.Dispose();
94     var minimizedInitialJson = Regex.Replace(initialJson,
95         ↪ "(\\(?:[^\\"\\\\]|\\\\\\\\.)*\\)|\\\\s+", "$1");
96     Assert.Equal(minimizedInitialJson, exportedJson);
97 }

```

```
91     }
92 }
```

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     public class JsonStorageTests
15     {
16         private readonly ITestOutputHelper output;
17         public static BalancedVariantConverter<TLink> BalancedVariantConverter;
18
19         public JsonStorageTests(ITestOutputHelper output)
20         {
21             this.output = output;
22         }
23
24         public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
25             ↪ Platform.IO.TemporaryFile());
26
27         public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
28         {
29             var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
30                 ↪ true);
31             return new UnitedMemoryLinks<TLink>(new
32                 ↪ FileMappedResizableDirectMemory(dataDBFilename),
33                 ↪ UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
34                 ↪ IndexTreeType.Default);
35         }
36
37         public static DefaultJsonStorage<TLink> CreateJsonStorage()
38         {
39             var links = CreateLinks();
40             return CreateJsonStorage(links);
41         }
42
43         public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links)
44         {
45             BalancedVariantConverter = new(links);
46             return new DefaultJsonStorage<TLink>(links, BalancedVariantConverter);
47         }
48
49         [Fact]
50         public void ConstructorsTest() => CreateJsonStorage();
51
52         [Fact]
53         public void CreateDocumentTest()
54         {
55             var defaultJsonStorage = CreateJsonStorage();
56             defaultJsonStorage.CreateDocument("documentName");
57         }
58
59         [Fact]
60         public void GetDocumentTest()
61         {
62             var defaultJsonStorage = CreateJsonStorage();
63             var createdDocumentLink = defaultJsonStorage.CreateDocument("documentName");
64             var foundDocumentLink = defaultJsonStorage.GetDocumentOrDefault("documentName");
65             Assert.Equal(createdDocumentLink, foundDocumentLink);
66         }
67
68         [Fact]
69         public void CreateObjectTest()
70         {
71             var defaultJsonStorage = CreateJsonStorage();
72             var object0 = defaultJsonStorage.CreateObjectValue();
73             var object1 = defaultJsonStorage.CreateObjectValue();
74             Assert.NotEqual(object0, object1);
75         }
76     }
77 }
```

```

72 [Fact]
73 public void CreateStringTest()
74 {
75     var defaultJsonStorage = CreateJsonStorage();
76     defaultJsonStorage.CreateString("string");
77 }
78
79 [Fact]
80 public void CreateMemberTest()
81 {
82     var defaultJsonStorage = CreateJsonStorage();
83     var document = defaultJsonStorage.CreateDocument("documentName");
84     defaultJsonStorage.AttachObject(document);
85     defaultJsonStorage.CreateMember("keyName");
86 }
87
88 [Fact]
89 public void AttachObjectValueToDocumentTest()
90 {
91     var links = CreateLinks();
92     var defaultJsonStorage = CreateJsonStorage(links);
93     TLink document = defaultJsonStorage.CreateDocument("documentName");
94     TLink documentValueLink = defaultJsonStorage.AttachObject(document);
95     TLink createdObjectValue = links.GetTarget(documentValueLink);
96
97     TLink valueMarker = links.GetSource(createdObjectValue);
98     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
99
100     TLink createdObject = links.GetTarget(createdObjectValue);
101     TLink objectMarker = links.GetSource(createdObject);
102     Assert.Equal(objectMarker, defaultJsonStorage.ObjectMarker);
103
104     TLink foundDocumentValue = defaultJsonStorage.GetValueLink(document);
105     Assert.Equal(createdObjectValue, foundDocumentValue);
106 }
107
108 [Fact]
109 public void AttachStringValueToDocumentTest()
110 {
111     var links = CreateLinks();
112     var defaultJsonStorage = CreateJsonStorage(links);
113     TLink document = defaultJsonStorage.CreateDocument("documentName");
114     TLink documentStringLink = defaultJsonStorage.AttachString(document, "stringName");
115     TLink createdStringValue = links.GetTarget(documentStringLink);
116
117     TLink valueMarker = links.GetSource(createdStringValue);
118     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
119
120     TLink createdString = links.GetTarget(createdStringValue);
121     TLink stringMarker = links.GetSource(createdString);
122     Assert.Equal(stringMarker, defaultJsonStorage.StringMarker);
123
124     TLink foundStringValue = defaultJsonStorage.GetValueLink(document);
125     Assert.Equal(createdStringValue, foundStringValue);
126 }
127
128 [Fact]
129 public void AttachNumberToDocumentTest()
130 {
131     var links = CreateLinks();
132     var defaultJsonStorage = CreateJsonStorage(links);
133     TLink document = defaultJsonStorage.CreateDocument("documentName");
134     TLink documentNumberLink = defaultJsonStorage.AttachNumber(document, 2021);
135     TLink createdNumberValue = links.GetTarget(documentNumberLink);
136
137     TLink valueMarker = links.GetSource(createdNumberValue);
138     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
139
140     TLink createdNumber = links.GetTarget(createdNumberValue);
141     TLink numberMarker = links.GetSource(createdNumber);
142     Assert.Equal(numberMarker, defaultJsonStorage.NumberMarker);
143
144     TLink foundNumberValue = defaultJsonStorage.GetValueLink(document);
145     Assert.Equal(createdNumberValue, foundNumberValue);
146 }
147
148 [Fact]
149 public void AttachTrueValueToDocumentTest()
150 {

```



```

151     var links = CreateLinks();
152     var defaultJsonStorage = CreateJsonStorage(links);
153     TLink document = defaultJsonStorage.CreateDocument("documentName");
154
155     TLink documentTrueValueLink = defaultJsonStorage.AttachBoolean(document, true);
156     TLink createdTrueValue = links.GetTarget(documentTrueValueLink);
157
158     TLink valueMarker = links.GetSource(createdTrueValue);
159     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
160
161     TLink trueMarker = links.GetTarget(createdTrueValue);
162     Assert.Equal(trueMarker, defaultJsonStorage.TrueMarker);
163
164     TLink foundTrueValue = defaultJsonStorage.GetValueLink(document);
165     Assert.Equal(createdTrueValue, foundTrueValue);
166 }
167
168 [Fact]
169 public void AttachFalseValueToDocumentTest()
170 {
171     var links = CreateLinks();
172     var defaultJsonStorage = CreateJsonStorage(links);
173     TLink document = defaultJsonStorage.CreateDocument("documentName");
174
175     TLink documentFalseValueLink = defaultJsonStorage.AttachBoolean(document, false);
176     TLink createdFalseValue = links.GetTarget(documentFalseValueLink);
177
178     TLink valueMarker = links.GetSource(createdFalseValue);
179     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
180
181     TLink falseMarker = links.GetTarget(createdFalseValue);
182     Assert.Equal(falseMarker, defaultJsonStorage.FalseMarker);
183
184     TLink foundFalseValue = defaultJsonStorage.GetValueLink(document);
185     Assert.Equal(createdFalseValue, foundFalseValue);
186 }
187
188 [Fact]
189 public void AttachNullValueToDocumentTest()
190 {
191     var links = CreateLinks();
192     var defaultJsonStorage = CreateJsonStorage(links);
193     TLink document = defaultJsonStorage.CreateDocument("documentName");
194
195     TLink documentNullValueLink = defaultJsonStorage.AttachNull(document);
196     TLink createdNullValue = links.GetTarget(documentNullValueLink);
197
198     TLink valueMarker = links.GetSource(createdNullValue);
199     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
200
201     TLink nullMarker = links.GetTarget(createdNullValue);
202     Assert.Equal(nullMarker, defaultJsonStorage.NullMarker);
203
204     TLink foundNullValue = defaultJsonStorage.GetValueLink(document);
205     Assert.Equal(createdNullValue, foundNullValue);
206 }
207
208 [Fact]
209 public void AttachEmptyArrayValueToDocumentTest()
210 {
211     var links = CreateLinks();
212     var defaultJsonStorage = CreateJsonStorage(links);
213     TLink document = defaultJsonStorage.CreateDocument("documentName");
214
215     TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, new
        ↪ TLink[0]);
216     TLink createdArrayValue = links.GetTarget(documentArrayValueLink);
217     output.WriteLine(links.Format(createdArrayValue));
218
219     TLink valueMarker = links.GetSource(createdArrayValue);
220     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
221
222     TLink createdArrayLink = links.GetTarget(createdArrayValue);
223     TLink arrayMarker = links.GetSource(createdArrayLink);
224     Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
225
226     TLink createArrayContents = links.GetTarget(createdArrayLink);
227     Assert.Equal(createArrayContents, defaultJsonStorage.EmptyArrayMarker);
228

```

```

229
230     TLink foundArrayValue = defaultJsonStorage.GetValueLink(document);
231     Assert.Equal(createdArrayValue, foundArrayValue);
232 }
233
234 [Fact]
235 public void AttachArrayValueToDocumentTest()
236 {
237     var links = CreateLinks();
238     var defaultJsonStorage = CreateJsonStorage(links);
239     TLink document = defaultJsonStorage.CreateDocument("documentName");
240
241     TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
242     TLink[] array = new TLink[] { arrayElement, arrayElement, arrayElement };
243
244
245     TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, array);
246     TLink createdArrayValue = links.GetTarget(documentArrayValueLink);
247
248     DefaultStack<TLink> stack = new();
249     RightSequenceWalker<TLink> rightSequenceWalker = new(links, stack, arrayElementLink
250         ↪ => links.GetSource(arrayElementLink) == defaultJsonStorage.ValueMarker);
251     IEnumerable<TLink> arrayElementsValuesLink =
252         ↪ rightSequenceWalker.Walk(createdArrayValue);
253     Assert.NotEmpty(arrayElementsValuesLink);
254
255     output.WriteLine(links.Format(createdArrayValue));
256
257     TLink valueMarker = links.GetSource(createdArrayValue);
258     Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
259
260     TLink createdArrayLink = links.GetTarget(createdArrayValue);
261     TLink arrayMarker = links.GetSource(createdArrayLink);
262     Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
263
264     TLink createdArrayContents = links.GetTarget(createdArrayLink);
265     Assert.Equal(links.GetTarget(createdArrayContents), arrayElement);
266
267     TLink foundArrayValue = defaultJsonStorage.GetValueLink(document);
268     Assert.Equal(createdArrayValue, foundArrayValue);
269 }
270
271 [Fact]
272 public void GetObjectFromDocumentObjectValueLinkTest()
273 {
274     ILinks<TLink> links = CreateLinks();
275     var defaultJsonStorage = CreateJsonStorage(links);
276     TLink document = defaultJsonStorage.CreateDocument("documentName");
277     TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
278     TLink objectValueLink = links.GetTarget(documentObjectValueLink);
279     TLink objectFromGetObject = defaultJsonStorage.GetObject(documentObjectValueLink);
280     output.WriteLine(links.Format(objectValueLink));
281     output.WriteLine(links.Format(objectFromGetObject));
282     Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
283 }
284
285 [Fact]
286 public void GetObjectFromObjectValueLinkTest()
287 {
288     ILinks<TLink> links = CreateLinks();
289     var defaultJsonStorage = CreateJsonStorage(links);
290     TLink document = defaultJsonStorage.CreateDocument("documentName");
291     TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
292     TLink objectValueLink = links.GetTarget(documentObjectValueLink);
293     TLink objectFromGetObject = defaultJsonStorage.GetObject(objectValueLink);
294     Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
295 }
296
297 [Fact]
298 public void AttachStringValueToKey()
299 {
300     ILinks<TLink> links = CreateLinks();
301     var defaultJsonStorage = CreateJsonStorage(links);
302     TLink document = defaultJsonStorage.CreateDocument("documentName");
303     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
304     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
305     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");

```

```

306 TLink memberStringValueLink = defaultJsonStorage.AttachString(memberLink,
    ↪ "stringValue");
307 TLink stringValueLink = links.GetTarget(memberStringValueLink);
308 List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
309 Assert.Equal(memberLink, objectMembersLinks[0]);
310 Assert.Equal(stringValueLink,
    ↪ defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
311 }
312
313 [Fact]
314 public void AttachNumberValueToKey()
315 {
316     ILinks<TLink> links = CreateLinks();
317     var defaultJsonStorage = CreateJsonStorage(links);
318     TLink document = defaultJsonStorage.CreateDocument("documentName");
319     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
320     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
321     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
322     TLink memberNumberValueLink = defaultJsonStorage.AttachNumber(memberLink, 123);
323     TLink numberValueLink = links.GetTarget(memberNumberValueLink);
324     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
325     Assert.Equal(memberLink, objectMembersLinks[0]);
326     Assert.Equal(numberValueLink,
        ↪ defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
327 }
328
329 [Fact]
330 public void AttachObjectValueToKey()
331 {
332     ILinks<TLink> links = CreateLinks();
333     var defaultJsonStorage = CreateJsonStorage(links);
334     TLink document = defaultJsonStorage.CreateDocument("documentName");
335     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
336     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
337     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
338     TLink memberObjectValueLink = defaultJsonStorage.AttachObject(memberLink);
339     TLink objectValueLink = links.GetTarget(memberObjectValueLink);
340     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
341     Assert.Equal(memberLink, objectMembersLinks[0]);
342     Assert.Equal(objectValueLink,
        ↪ defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
343 }
344
345 [Fact]
346 public void AttachArrayValueToKey()
347 {
348     ILinks<TLink> links = CreateLinks();
349     var defaultJsonStorage = CreateJsonStorage(links);
350     TLink document = defaultJsonStorage.CreateDocument("documentName");
351     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
352     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
353     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
354     TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
355     TLink[] array = { arrayElement, arrayElement, arrayElement };
356     TLink memberArrayValueLink = defaultJsonStorage.AttachArray(memberLink, array);
357     TLink arrayValueLink = links.GetTarget(memberArrayValueLink);
358     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
359     Assert.Equal(memberLink, objectMembersLinks[0]);
360     Assert.Equal(arrayValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
361 }
362
363 [Fact]
364 public void AttachTrueValueToKey()
365 {
366     ILinks<TLink> links = CreateLinks();
367     var defaultJsonStorage = CreateJsonStorage(links);
368     TLink document = defaultJsonStorage.CreateDocument("documentName");
369     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
370     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
371     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
372     TLink memberTrueValueLink = defaultJsonStorage.AttachBoolean(memberLink, true);
373     TLink trueValueLink = links.GetTarget(memberTrueValueLink);
374     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
375     Assert.Equal(memberLink, objectMembersLinks[0]);
376     Assert.Equal(trueValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
377 }
378
379 [Fact]

```

```

380 public void AttachFalseValueToKey()
381 {
382     ILinks<TLink> links = CreateLinks();
383     var defaultJsonStorage = CreateJsonStorage(links);
384     TLink document = defaultJsonStorage.CreateDocument("documentName");
385     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
386     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
387     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
388     TLink memberFalseValueLink = defaultJsonStorage.AttachBoolean(memberLink, false);
389     TLink falseValueLink = links.GetTarget(memberFalseValueLink);
390     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
391     Assert.Equal(memberLink, objectMembersLinks[0]);
392     Assert.Equal(falseValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
393 }
394
395 [Fact]
396 public void AttachNullValueToKey()
397 {
398     ILinks<TLink> links = CreateLinks();
399     var defaultJsonStorage = CreateJsonStorage(links);
400     TLink document = defaultJsonStorage.CreateDocument("documentName");
401     TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
402     TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
403     TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
404     TLink memberNullValueLink = defaultJsonStorage.AttachNull(memberLink);
405     TLink nullValueLink = links.GetTarget(memberNullValueLink);
406     List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
407     Assert.Equal(nullValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
408 }
409 }
410 }

```

Index

./csharp/Platform.Data.Doublets.Json.Tests/JsonImportAndExportTests.cs, 13
./csharp/Platform.Data.Doublets.Json.Tests/JsonStorageTests.cs, 15
./csharp/Platform.Data.Doublets.Json/DefaultJsonStorage.cs, 1
./csharp/Platform.Data.Doublets.Json/IJsonStorage.cs, 6
./csharp/Platform.Data.Doublets.Json/JsonArrayElementCriterionMatcher.cs, 6
./csharp/Platform.Data.Doublets.Json/JsonExporter.cs, 7
./csharp/Platform.Data.Doublets.Json/JsonExporterCli.cs, 9
./csharp/Platform.Data.Doublets.Json/JsonImporter.cs, 10
./csharp/Platform.Data.Doublets.Json/JsonImporterCli.cs, 12