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

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

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

namespace Platform.Data.Doublets.Json
{
    public class DefaultJsonStorage<TLink> : IJsonStorage<TLink>
        where TLink : struct
    {
        public readonly TLink Any;
        public static readonly TLink Zero = default;
        public static readonly TLink One = Arithmetic.Increment(Zero);
        public readonly BalancedVariantConverter<TLink> BalancedVariantConverter;
        public readonly IConverter<IList<TLink>, TLink> ListToSequenceConverter;
        public readonly TLink MeaningRoot;
        public readonly EqualityComparer<TLink> EqualityComparer =
        ↪  EqualityComparer<TLink>.Default;
        // Converters that are able to convert link's address (UInt64 value) to a raw number
        ↪  represented with another UInt64 value and back
        public readonly RawNumberToAddressConverter<TLink> NumberToAddressConverter = new();
        public readonly AddressToRawNumberConverter<TLink> AddressToNumberConverter = new();
        // Converters between BigInteger and raw number sequence
        public readonly BigIntegerToRawNumberSequenceConverter<TLink>
        ↪  BigIntegerToRawNumberSequenceConverter;
        public readonly RawNumberSequenceToBigIntegerConverter<TLink>
        ↪  RawNumberSequenceToBigIntegerConverter;
        // Converters between decimal and rational number sequence
        public readonly DecimalToRationalConverter<TLink> DecimalToRationalConverter;
        public readonly RationalToDecimalConverter<TLink> RationalToDecimalConverter;
        // Converters between string and unicode sequence
        public readonly IConverter<string, TLink> StringToUnicodeSequenceConverter;
        public readonly IConverter<TLink, string> UnicodeSequenceToStringConverter;
        // For sequences
        public readonly JsonArrayElementCriterionMatcher<TLink> JsonArrayElementCriterionMatcher;
        public readonly DefaultSequenceRightHeightProvider<TLink>
        ↪  DefaultSequenceRightHeightProvider;
        public readonly DefaultSequenceAppender<TLink> DefaultSequenceAppender;
        public ILinks<TLink> Links { get; }
        public TLink DocumentMarker { get; }
        public TLink ObjectMarker { get; }
        public TLink MemberMarker { get; }
        public TLink ValueMarker { get; }
        public TLink StringMarker { get; }
        public TLink EmptyStringMarker { get; }
        public TLink NumberMarker { get; }
        public TLink NegativeNumberMarker { get; }
        public TLink ArrayMarker { get; }
        public TLink EmptyArrayMarker { get; }
        public TLink TrueMarker { get; }
        public TLink FalseMarker { get; }
        public TLink NullMarker { get; }

        public DefaultJsonStorage(ILinks<TLink> links, IConverter<IList<TLink>, TLink>
        ↪  listToSequenceConverter)
        {
            Links = links;
            ListToSequenceConverter = listToSequenceConverter;
            // Initializes constants
            Any = Links.Constants.Any;
            var markerIndex = One;
            MeaningRoot = links.GetOrCreate(markerIndex, markerIndex);
            var unicodeSymbolMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
            ↪  markerIndex));
            var unicodeSequenceMarker = links.GetOrCreate(MeaningRoot, Arithmetic.Increment(ref
            ↪  markerIndex));
```

```
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         {
```

```
            var documentName = CreateString(name);
            return Links.GetOrCreate(DocumentMarker, documentName);
        }

        public TLink CreateObject()
        {
            var @object = Links.Create();
            return Links.Update(@object, newSource: ObjectMarker, newTarget: @object);
        }

        public TLink CreateObjectValue()
        {
            var @object = CreateObject();
            return CreateValue(@object);
        }

        public TLink CreateArray(IList<TLink> array)
        {
            var arraySequence = array.Count == 0 ? EmptyArrayMarker :
            ↪  BalancedVariantConverter.Convert(array);
            return CreateArray(arraySequence);
        }

        public TLink CreateArray(TLink sequence) => Links.GetOrCreate(ArrayMarker, sequence);

        public TLink CreateArrayValue(IList<TLink> array)
        {
            var arrayLink = CreateArray(array);
            return CreateValue(arrayLink);
        }

        public TLink CreateArrayValue(TLink sequence)
        {
            var array = CreateArray(sequence);
            return CreateValue(array);
        }

        public TLink CreateMember(string name)
        {
            var nameLink = CreateString(name);
            return Links.GetOrCreate(MemberMarker, nameLink);
        }

        public TLink CreateValue(TLink value) => Links.GetOrCreate(ValueMarker, value);

        public TLink AttachObject(TLink parent) => Attach(parent, CreateObjectValue());

        public TLink AttachString(TLink parent, string content)
        {
            var @string = CreateString(content);
            var stringValue = CreateValue(@string);
            return Attach(parent, stringValue);
        }

        public TLink AttachNumber(TLink parent, decimal number)
        {
            var numberLink = CreateNumber(number);
            var numberValue = CreateValue(numberLink);
            return Attach(parent, numberValue);
        }

        public TLink AttachBoolean(TLink parent, bool value)
        {
            var booleanValue = CreateBooleanValue(value);
            return Attach(parent, booleanValue);
        }

        public TLink AttachNull(TLink parent)
        {
            var nullValue = CreateNullValue();
            return Attach(parent, nullValue);
        }

        public TLink AttachArray(TLink parent, IList<TLink> array)
        {
            var arrayValue = CreateArrayValue(array);
            return Attach(parent, arrayValue);
        }

```

```
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 
292         public TLink GetObject(TLink objectValueLink)
293         {
294             var current = objectValueLink;
```

```
            TLink source;
            for (int i = 0; i < 3; i++)
            {
                source = Links.GetSource(current);
                if (EqualityComparer.Equals(source, ObjectMarker))
                {
                    return current;
                }
                current = Links.GetTarget(current);
            }
            throw new Exception("The passed link does not contain an object.");
        }

        public TLink GetArray(TLink arrayValueLink)
        {
            var current = arrayValueLink;
            TLink source;
            for (int i = 0; i < 3; i++)
            {
                source = Links.GetSource(current);
                if (EqualityComparer.Equals(source, ArrayMarker))
                {
                    return current;
                }
                current = Links.GetTarget(current);
            }
            throw new Exception("The passed link does not contain an array.");
        }

        public TLink GetArraySequence(TLink array) => Links.GetTarget(array);

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

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

        public List<TLink> GetMembersLinks(TLink @object)
        {
            Link<TLink> query = new(index: Any, source: @object, target: Any);
            List<TLink> members = new();
            Links.Each(objectMemberLink =>
            {
                var memberLink = Links.GetTarget(objectMemberLink);
                var memberMarker = Links.GetSource(memberLink);
                if (EqualityComparer.Equals(memberMarker, MemberMarker))
                {
                    members.Add(Links.GetIndex(objectMemberLink));
                }
```

```
                return Links.Constants.Continue;
            }, query);
            return members;
        }
    }
}
```

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

```
using System.Collections.Generic;

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

namespace Platform.Data.Doublets.Json
{
    public interface IJsonStorage<TLink>
    {
        public ILinks<TLink> Links { get; }
        public TLink DocumentMarker { get; }
        public TLink ObjectMarker { get; }
        public TLink StringMarker { get; }
        public TLink EmptyStringMarker { get; }
        public TLink MemberMarker { get; }
        public TLink ValueMarker { get; }
        public TLink NumberMarker { get; }
        public TLink ArrayMarker { get; }
        public TLink EmptyArrayMarker { get; }
        public TLink TrueMarker { get; }
        public TLink FalseMarker { get; }
        public TLink NullMarker { get; }
        TLink CreateString(string content);
        TLink CreateStringValue(string content);
        TLink CreateNumber(decimal number);
        TLink CreateNumberValue(decimal number);
        TLink CreateBooleanValue(bool value);
        TLink CreateNullValue();
        TLink CreateDocument(string name);
        TLink GetDocumentOrDefault(string name);
        TLink CreateObject();
        TLink CreateObjectValue();
        TLink CreateArray(IList<TLink> array);
        TLink CreateArrayValue(IList<TLink> array) => CreateValue(CreateArray(array));
        TLink CreateArrayValue(TLink array) => CreateValue(array);
        TLink CreateMember(string name);
        TLink CreateValue(TLink value);
        TLink Attach(TLink source, TLink target);
        TLink AttachObject(TLink parent);
        TLink AttachString(TLink parent, string content);
        TLink AttachNumber(TLink parent, decimal number);
        TLink AttachBoolean(TLink parent, bool value);
        TLink AttachNull(TLink parent);
        TLink AttachArray(TLink parent, IList<TLink> array);
        TLink AttachMemberToObject(TLink @object, string keyName);
        TLink AppendArrayValue(TLink arrayValue, TLink appendant);
        string GetString(TLink stringValue);
        decimal GetNumber(TLink value);
        TLink GetObject(TLink objectValue);
        TLink GetArray(TLink arrayValueLink);
        TLink GetArraySequence(TLink array);
        TLink GetValueLink(TLink parent);
        TLink GetValueMarker(TLink link);
        List<TLink> GetMembersLinks(TLink @object);
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Text.Json;
using System.Threading;
using System.IO;
using Platform.Converters;
using System.Collections;
using Platform.Data.Doublets.Sequences;
using Platform.Data.Doublets.Sequences.HeightProviders;
using Platform.Data.Doublets.Sequences.CriterionMatchers;
using Platform.Interfaces;
```

```

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

namespace Platform.Data.Doublets.Json
{
    public class JsonArrayElementCriterionMatcher<TLink> : ICriterionMatcher<TLink>
    {
        public readonly IJsonStorage<TLink> Storage;
        public JsonArrayElementCriterionMatcher(IJsonStorage<TLink> storage) => Storage =
        ↪  storage;
        public bool IsMatched(TLink link) =>
        ↪  EqualityComparer<TLink>.Default.Equals(Storage.Links.GetSource(link),
        ↪  Storage.ValueMarker);
    }
}
```

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

```
using System;
using System.Collections.Generic;
using System.Text.Json;
using System.Threading;
using Platform.Data.Doublets.Sequences.Walkers;
using Platform.Collections.Stacks;

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

namespace Platform.Data.Doublets.Json
{
    public class JsonExporter<TLink>
    {
        public readonly IJsonStorage<TLink> Storage;
        public readonly EqualityComparer<TLink> EqualityComparer =
        ↪  EqualityComparer<TLink>.Default;

        public JsonExporter(IJsonStorage<TLink> storage) => Storage = storage;

            private bool IsElement(TLink link)
        {
            var marker = Storage.Links.GetSource(link);
            return EqualityComparer.Equals(marker, Storage.ValueMarker);
        }

        private void WriteStringValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
        ↪  utf8JsonWriter.WriteStringValue(Storage.GetString(valueLink));

        private void WriteString(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
        ↪  valueLink) => utf8JsonWriter.WriteString(parent, Storage.GetString(valueLink));

        private void WriteNumberValue(in Utf8JsonWriter utf8JsonWriter, TLink valueLink) =>
        ↪  utf8JsonWriter.WriteNumberValue(Storage.GetNumber(valueLink));

        private void WriteNumber(in Utf8JsonWriter utf8JsonWriter, string parent, TLink
        ↪  valueLink) => utf8JsonWriter.WriteNumber(parent, Storage.GetNumber(valueLink));

        private void Write(ref Utf8JsonWriter utf8JsonWriter, string parent, TLink valueLink,
        ↪  CancellationToken cancellationToken)
        {
            if (cancellationToken.IsCancellationRequested)
            {
                return;
            }
            var valueMarker = Storage.GetValueMarker(valueLink);
            if (EqualityComparer.Equals(valueMarker, Storage.ObjectMarker))
            {
                utf8JsonWriter.WriteStartObject(parent);
                var membersLinks = Storage.GetMembersLinks(Storage.GetObject(valueLink));
                foreach (var memberLink in membersLinks)
                {
                    if (cancellationToken.IsCancellationRequested)
                    {
                        return;
                    }
                    Write(ref utf8JsonWriter, Storage.GetString(memberLink),
                    ↪  Storage.GetValueLink(memberLink), cancellationToken);
                }
                utf8JsonWriter.WriteEndObject();
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.ArrayMarker))
            {
```

```
                var array = Storage.GetArray(valueLink);
                var sequence = Storage.GetArraySequence(array);
                utf8JsonWriter.WriteStartArray(parent);
                if (!EqualityComparer.Equals(sequence, Storage.EmptyArrayMarker))
                {
                    RightSequenceWalker<TLink> rightSequenceWalker = new(Storage.Links, new
                    ↪  DefaultStack<TLink>(), IsElement);
                    var elements = rightSequenceWalker.Walk(sequence);
                    foreach (var element in elements)
                    {
                        if (cancellationToken.IsCancellationRequested)
                        {
                            return;
                        }
                        Write(ref utf8JsonWriter, element, in cancellationToken);
                    }
                }
                utf8JsonWriter.WriteEndArray();
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.StringMarker))
            {
                WriteString(in utf8JsonWriter, parent, valueLink);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.NumberMarker))
            {
                WriteNumber(in utf8JsonWriter, parent, valueLink);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
            {
                utf8JsonWriter.WriteBoolean(parent, true);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
            {
                utf8JsonWriter.WriteBoolean(parent, false);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
            {
                utf8JsonWriter.WriteNull(parent);
            }
        }

        private void Write(ref Utf8JsonWriter utf8JsonWriter, TLink valueLink, in
        ↪  CancellationToken cancellationToken)
        {
            if (cancellationToken.IsCancellationRequested)
            {
                return;
            }
            var valueMarker = Storage.GetValueMarker(valueLink);
            if (EqualityComparer.Equals(valueMarker, Storage.ObjectMarker))
            {
                utf8JsonWriter.WriteStartObject();
                var membersLinks = Storage.GetMembersLinks(Storage.GetObject(valueLink));
                foreach (var memberLink in membersLinks)
                {
                    if (cancellationToken.IsCancellationRequested)
                    {
                        return;
                    }
                    Write(ref utf8JsonWriter, Storage.GetString(memberLink),
                    ↪  Storage.GetValueLink(memberLink), cancellationToken);
                }
                utf8JsonWriter.WriteEndObject();
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.ArrayMarker))
            {
                var array = Storage.GetArray(valueLink);
                var sequence = Storage.GetArraySequence(array);
                utf8JsonWriter.WriteStartArray();
                if (!EqualityComparer.Equals(sequence, Storage.EmptyArrayMarker))
                {
                    RightSequenceWalker<TLink> rightSequenceWalker = new(Storage.Links, new
                    ↪  DefaultStack<TLink>(), IsElement);
                    var elements = rightSequenceWalker.Walk(sequence);
                    foreach (var element in elements)
                    {
                        if (cancellationToken.IsCancellationRequested)
                        {
```

```
                            return;
                        }
                        Write(ref utf8JsonWriter, element, in cancellationToken);
                    }
                }
                utf8JsonWriter.WriteEndArray();
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.StringMarker))
            {
                WriteStringValue(in utf8JsonWriter, valueLink);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.NumberMarker))
            {
                WriteNumberValue(in utf8JsonWriter, valueLink);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.TrueMarker))
            {
                utf8JsonWriter.WriteBooleanValue(true);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.FalseMarker))
            {
                utf8JsonWriter.WriteBooleanValue(false);
            }
            else if (EqualityComparer.Equals(valueMarker, Storage.NullMarker))
            {
                utf8JsonWriter.WriteNullValue();
            }
        }

        public void Export(TLink document, ref Utf8JsonWriter utf8JsonWriter, in
        ↪  CancellationToken cancellationToken)
        {
            if (EqualityComparer.Equals(document, default))
            {
                throw new Exception("No document with this name exists");
            }
            var valueLink = Storage.GetValueLink(document);
            Write(ref utf8JsonWriter, valueLink, in cancellationToken);
            utf8JsonWriter.Flush();
        }

        public void Export(string documentName, Utf8JsonWriter utf8JsonWriter, CancellationToken
        ↪  cancellationToken) => Export(Storage.GetDocumentOrDefault(documentName), ref
        ↪  utf8JsonWriter, in cancellationToken);
    }
}
```

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

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

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

namespace Platform.Data.Doublets.Json
{
    public class JsonExporterCli<TLink>
        where TLink : struct
    {
        public void Run(params string[] args)
        {
            var linksFilePath = ConsoleHelpers.GetOrReadArgument(0, "Links file path", args);
            var jsonFilePath = ConsoleHelpers.GetOrReadArgument(1, "JSON file path", args);
            var documentName = ConsoleHelpers.GetOrReadArgument(2, "Document name", args);
            if (!File.Exists(linksFilePath))
            {
                Console.WriteLine($"${linksFilePath} file does not exist.");
            }
            using FileStream jsonFileStream = new(jsonFilePath, FileMode.Append);
            JsonWriterOptions utf8JsonWriterOptions = new()
            {
                Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping,
                Indented = true
```

```
            };
            Utf8JsonWriter utf8JsonWriter = new(jsonFileStream, utf8JsonWriterOptions);
            var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
            ↪  true);
            using UnitedMemoryLinks<TLink> memoryAdapter = new (new
            ↪  FileMappedResizableDirectMemory(linksFilePath),
            ↪  UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
            ↪  IndexTreeType.Default);
            var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
            BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
            var storage = new DefaultJsonStorage<TLink>(links, balancedVariantConverter);
            var exporter = new JsonExporter<TLink>(storage);
            var document = storage.GetDocumentOrDefault(documentName);
            if (storage.EqualityComparer.Equals(document, default))
            {
                Console.WriteLine("No document with this name.");
            }
            using ConsoleCancellation cancellation = new ();
            var cancellationToken = cancellation.Token;
            Console.WriteLine("Press CTRL+C to stop.");
            try
            {
                exporter.Export(document, ref utf8JsonWriter, in cancellationToken);
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception);
                return;
            }
            finally
            {
                utf8JsonWriter.Dispose();
            }
            Console.WriteLine("Export completed successfully.");
        }
    }
}
```

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

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

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

namespace Platform.Data.Doublets.Json
{
    public class JsonImporter<TLink>
    {
        public readonly IJsonStorage<TLink> Storage;
        public readonly EqualityComparer<TLink> EqualityComparer =
        ↪  EqualityComparer<TLink>.Default;
        public readonly Stack<TLink> Parents = new ();
        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();
            }
        }

        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();
                        if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                        {
                            Parents.Pop();
                            newParentArray = Storage.AppendArrayValue(parent, value);
                            Parents.Push(newParentArray);
                            Parents.Push(value);
                        }
                        else
                        {
                            var @object = Storage.Attach(parent, value);
                            Parents.Push(@object);
                        }
                        break;
                    }
                    case JsonTokenType.EndObject:
                        Parents.Pop();
                        break;
                    case JsonTokenType.StartArray:
                        value = Storage.CreateArrayValue(Array.Empty<TLink>());
                        Parents.Push(value);
                        break;
                    case JsonTokenType.EndArray:
                    {
                        var arrayValue = Parents.Pop();
                        parent = Parents.Peek();
                        parentMarker = Storage.GetValueMarker(parent);
                        if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                        {
                            Parents.Pop();
                            newParentArray = Storage.AppendArrayValue(parent, arrayValue);
                            Parents.Push(newParentArray);
                        }
                        Storage.Attach(parent, arrayValue);
                        break;
                    }
                    case JsonTokenType.String:
                    {
                        var @string = utf8JsonReader.GetString();
                        value = Storage.CreateStringValue(@string);
                        if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                        {
                            Parents.Pop();
                            newParentArray = Storage.AppendArrayValue(parent, value);
                            Parents.Push(newParentArray);
                        }
                        else
                        {
                            Storage.Attach(parent, value);
                        }
                        break;
                    }
                    case JsonTokenType.Number:
                    {
                        value = Storage.CreateNumberValue(utf8JsonReader.GetDecimal());
                        if (EqualityComparer.Equals(parentMarker, Storage.ArrayMarker))
                        {
                            Parents.Pop();
                            newParentArray = Storage.AppendArrayValue(parent, value);
                            Parents.Push(newParentArray);
                        }
```

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

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

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

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

namespace Platform.Data.Doublets.Json
{
    public class JsonImporterCli<TLink>
        where TLink : struct
```

```
    {
        public void Run(params string[] args)
        {
            var jsonFilePath = ConsoleHelpers.GetOrReadArgument(0, "JSON file path", args);
            var documentName = ConsoleHelpers.GetOrReadArgument(1, "Document name", args);
            var linksFilePath = ConsoleHelpers.GetOrReadArgument(2, "Links file path", args);
            if (!File.Exists(jsonFilePath))
            {
                Console.WriteLine($"${jsonFilePath} file does not exist.");
            }
            var json = File.ReadAllText(jsonFilePath);
            var encodedJson = Encoding.UTF8.GetBytes(json);
            ReadOnlySpan<byte> readOnlySpanEncodedJson = new(encodedJson);
            Utf8JsonReader utf8JsonReader = new(readOnlySpanEncodedJson);
            LinksConstants<TLink> linksConstants = new(enableExternalReferencesSupport: true);
            FileMappedResizableDirectMemory fileMappedResizableDirectMemory = new(linksFilePath);
            var unitedMemoryLinks = UnitedMemoryLinks<TLink>.DefaultLinksSizeStep;
            const IndexTreeType indexTreeType = IndexTreeType.Default;
            using UnitedMemoryLinks<TLink> memoryAdapter = new(fileMappedResizableDirectMemory,
            ↪  unitedMemoryLinks, linksConstants, indexTreeType);
            var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
            BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
            DefaultJsonStorage<TLink> storage = new(links, balancedVariantConverter);
            JsonImporter<TLink> importer = new(storage);
            using ConsoleCancellation cancellation = new();
            var cancellationToken = cancellation.Token;
            Console.WriteLine("Press CTRL+C to stop.");
            try
            {
                importer.Import(documentName, ref utf8JsonReader, in cancellationToken);
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception);
                return;
            }
            Console.WriteLine("Import completed successfully.");
        }
    }
}
```

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

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

namespace Platform.Data.Doublets.Json.Tests
{
    public class JsonImportAndExportTests
    {
        public static BalancedVariantConverter<TLink> BalancedVariantConverter;

        public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new IO.TemporaryFile());

        public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
        {
            var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
            ↪  true);
            return new UnitedMemoryLinks<TLink>(new
            ↪  FileMappedResizableDirectMemory(dataDBFilename),
            ↪  UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
            ↪  IndexTreeType.Default);
        }

        public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links) => new
        ↪  (links, BalancedVariantConverter);

        public TLink Import(IJsonStorage<TLink> storage, string documentName, byte[] json)
        {
            Utf8JsonReader utf8JsonReader = new(json);
            JsonImporter<TLink> jsonImporter = new(storage);
```

```
            CancellationTokenSource importCancellationTokenSource = new();
            CancellationToken cancellationToken = importCancellationTokenSource.Token;
            return jsonImporter.Import(documentName, ref utf8JsonReader, in cancellationToken);
        }

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

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

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

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

namespace Platform.Data.Doublets.Json.Tests
{
    public class JsonStorageTests
```

```
16      {
17          private readonly ITestOutputHelper output;
18          public static BalancedVariantConverter<TLink> BalancedVariantConverter;
19
20          public JsonStorageTests(ITestOutputHelper output)
21          {
22              this.output = output;
23          }
24
25          public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
            ↪  Platform.IO.TemporaryFile());
26
27          public static ILinks<TLink> CreateLinks<TLink>(string dataDBFilename)
28          {
29              var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
                ↪  true);
30              return new UnitedMemoryLinks<TLink>(new
                ↪  FileMappedResizableDirectMemory(dataDBFilename),
                ↪  UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
                ↪  IndexTreeType.Default);
31          }
32
33          public static DefaultJsonStorage<TLink> CreateJsonStorage()
34          {
35              var links = CreateLinks();
36              return CreateJsonStorage(links);
37          }
38
39          public static DefaultJsonStorage<TLink> CreateJsonStorage(ILinks<TLink> links)
40          {
41              BalancedVariantConverter = new(links);
42              return new DefaultJsonStorage<TLink>(links, BalancedVariantConverter);
43          }
44
45          [Fact]
46          public void ConstructorsTest() => CreateJsonStorage();
47
48          [Fact]
49          public void CreateDocumentTest()
50          {
51              var defaultJsonStorage = CreateJsonStorage();
52              defaultJsonStorage.CreateDocument("documentName");
53          }
54
55          [Fact]
56          public void GetDocumentTest()
57          {
58              var defaultJsonStorage = CreateJsonStorage();
59              var createdDocumentLink = defaultJsonStorage.CreateDocument("documentName");
60              var foundDocumentLink = defaultJsonStorage.GetDocumentOrDefault("documentName");
61              Assert.Equal(createdDocumentLink, foundDocumentLink);
62          }
63
64          [Fact]
65          public void CreateObjectTest()
66          {
67              var defaultJsonStorage = CreateJsonStorage();
68              var object0 = defaultJsonStorage.CreateObjectValue();
69              var object1 = defaultJsonStorage.CreateObjectValue();
70              Assert.NotEqual(object0, object1);
71          }
72
73          [Fact]
74          public void CreateStringTest()
75          {
76              var defaultJsonStorage = CreateJsonStorage();
77              defaultJsonStorage.CreateString("string");
78          }
79
80          [Fact]
81          public void CreateMemberTest()
82          {
83              var defaultJsonStorage = CreateJsonStorage();
84              var document = defaultJsonStorage.CreateDocument("documentName");
85              defaultJsonStorage.AttachObject(document);
86              defaultJsonStorage.CreateMember("keyName");
87          }
88
89          [Fact]
```

```
90          public void AttachObjectValueToDocumentTest()
91          {
92              var links = CreateLinks();
93              var defaultJsonStorage =CreateJsonStorage(links);
94              TLink document = defaultJsonStorage.CreateDocument("documentName");
95              TLink documentValueLink = defaultJsonStorage.AttachObject(document);
96              TLink createdObjectValue = links.GetTarget(documentValueLink);
97
98              TLink valueMarker = links.GetSource(createdObjectValue);
99              Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
100
101             TLink createdObject = links.GetTarget(createdObjectValue);
102             TLink objectMarker = links.GetSource(createdObject);
103             Assert.Equal(objectMarker, defaultJsonStorage.ObjectMarker);
104
105             TLink foundDocumentValue = defaultJsonStorage.GetValueLink(document);
106             Assert.Equal(createdObjectValue, foundDocumentValue);
107         }
108
109         [Fact]
110         public void AttachStringValueToDocumentTest()
111         {
112             var links = CreateLinks();
113             var defaultJsonStorage =CreateJsonStorage(links);
114             TLink document = defaultJsonStorage.CreateDocument("documentName");
115             TLink documentStringLink = defaultJsonStorage.AttachString(document, "stringName");
116             TLink createdStringValue = links.GetTarget(documentStringLink);
117
118             TLink valueMarker = links.GetSource(createdStringValue);
119             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
120
121             TLink createdString = links.GetTarget(createdStringValue);
122             TLink stringMarker = links.GetSource(createdString);
123             Assert.Equal(stringMarker, defaultJsonStorage.StringMarker);
124
125             TLink foundStringValue = defaultJsonStorage.GetValueLink(document);
126             Assert.Equal(createdStringValue, foundStringValue);
127         }
128
129         [Fact]
130         public void AttachNumberToDocumentTest()
131         {
132             var links = CreateLinks();
133             var defaultJsonStorage = CreateJsonStorage(links);
134             TLink document = defaultJsonStorage.CreateDocument("documentName");
135             TLink documentNumberLink = defaultJsonStorage.AttachNumber(document, 2021);
136             TLink createdNumberValue = links.GetTarget(documentNumberLink);
137
138             TLink valueMarker = links.GetSource(createdNumberValue);
139             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
140
141             TLink createdNumber = links.GetTarget(createdNumberValue);
142             TLink numberMarker = links.GetSource(createdNumber);
143             Assert.Equal(numberMarker, defaultJsonStorage.NumberMarker);
144
145             TLink foundNumberValue = defaultJsonStorage.GetValueLink(document);
146             Assert.Equal(createdNumberValue, foundNumberValue);
147         }
148
149         [Fact]
150         public void AttachTrueValueToDocumentTest()
151         {
152             var links = CreateLinks();
153             var defaultJsonStorage =CreateJsonStorage(links);
154             TLink document = defaultJsonStorage.CreateDocument("documentName");
155
156             TLink documentTrueValueLink = defaultJsonStorage.AttachBoolean(document, true);
157             TLink createdTrueValue = links.GetTarget(documentTrueValueLink);
158
159             TLink valueMarker = links.GetSource(createdTrueValue);
160             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
161
162             TLink trueMarker = links.GetTarget(createdTrueValue);
163             Assert.Equal(trueMarker, defaultJsonStorage.TrueMarker);
164
165             TLink foundTrueValue = defaultJsonStorage.GetValueLink(document);
166             Assert.Equal(createdTrueValue, foundTrueValue);
167         }
168
```

```
169         [Fact]
170         public void AttachFalseValueToDocumentTest()
171         {
172             var links = CreateLinks();
173             var defaultJsonStorage =CreateJsonStorage(links);
174             TLink document = defaultJsonStorage.CreateDocument("documentName");
175 
176             TLink documentFalseValueLink = defaultJsonStorage.AttachBoolean(document, false);
177             TLink createdFalseValue = links.GetTarget(documentFalseValueLink);
178 
179             TLink valueMarker = links.GetSource(createdFalseValue);
180             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
181 
182             TLink falseMarker = links.GetTarget(createdFalseValue);
183             Assert.Equal(falseMarker, defaultJsonStorage.FalseMarker);
184 
185             TLink foundFalseValue = defaultJsonStorage.GetValueLink(document);
186             Assert.Equal(createdFalseValue, foundFalseValue);
187         }
188 
189         [Fact]
190         public void AttachNullValueToDocumentTest()
191         {
192             var links = CreateLinks();
193             var defaultJsonStorage =CreateJsonStorage(links);
194             TLink document = defaultJsonStorage.CreateDocument("documentName");
195 
196             TLink documentNullValueLink = defaultJsonStorage.AttachNull(document);
197             TLink createdNullValue = links.GetTarget(documentNullValueLink);
198 
199             TLink valueMarker = links.GetSource(createdNullValue);
200             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
201 
202             TLink nullMarker = links.GetTarget(createdNullValue);
203             Assert.Equal(nullMarker, defaultJsonStorage.NullMarker);
204 
205             TLink foundNullValue = defaultJsonStorage.GetValueLink(document);
206             Assert.Equal(createdNullValue, foundNullValue);
207         }
208 
209         [Fact]
210         public void AttachEmptyArrayValueToDocumentTest()
211         {
212             var links = CreateLinks();
213             var defaultJsonStorage =CreateJsonStorage(links);
214             TLink document = defaultJsonStorage.CreateDocument("documentName");
215 
216             TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, new
                ↪  TLink[0]);
217             TLink createdArrayValue = links.GetTarget(documentArrayValueLink);
218             output.WriteLine(links.Format(createdArrayValue));
219 
220 
221             TLink valueMarker = links.GetSource(createdArrayValue);
222             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
223 
224             TLink createdArrayLink = links.GetTarget(createdArrayValue);
225             TLink arrayMarker = links.GetSource(createdArrayLink);
226             Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
227 
228             TLink createArrayContents = links.GetTarget(createdArrayLink);
229             Assert.Equal(createArrayContents, defaultJsonStorage.EmptyArrayMarker);
230 
231             TLink foundArrayValue = defaultJsonStorage.GetValueLink(document);
232             Assert.Equal(createdArrayValue, foundArrayValue);
233         }
234 
235         [Fact]
236         public void AttachArrayValueToDocumentTest()
237         {
238             var links = CreateLinks();
239             var defaultJsonStorage =CreateJsonStorage(links);
240             TLink document = defaultJsonStorage.CreateDocument("documentName");
241 
242             TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
243             TLink[] array = new TLink[3] { arrayElement, arrayElement, arrayElement };
244 
245 
246             TLink documentArrayValueLink = defaultJsonStorage.AttachArray(document, array);
```

```
247             TLink createdArrayValue = links.GetTarget(documentArrayValueLink);
248
249             DefaultStack<TLink> stack = new();
250             RightSequenceWalker<TLink> rightSequenceWalker = new(links, stack, (TLink
                ↪  arrayElementLink) => links.GetSource(arrayElementLink) ==
                ↪  defaultJsonStorage.ValueMarker);
251             IEnumerable<TLink> arrayElementsValuesLink =
                ↪  rightSequenceWalker.Walk(createdArrayValue);
252             Assert.NotEmpty(arrayElementsValuesLink);
253
254             output.WriteLine(links.Format(createdArrayValue));
255
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
268             TLink foundArrayValue = defaultJsonStorage.GetValueLink(document);
269             Assert.Equal(createdArrayValue, foundArrayValue);
270         }
271
272         [Fact]
273         public void GetObjectFromDocumentObjectValueLinkTest()
274         {
275             ILinks<TLink> links = CreateLinks();
276             var defaultJsonStorage =CreateJsonStorage(links);
277             TLink document = defaultJsonStorage.CreateDocument("documentName");
278             TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
279             TLink objectValueLink = links.GetTarget(documentObjectValueLink);
280             TLink objectFromGetObject = defaultJsonStorage.GetObject(documentObjectValueLink);
281             output.WriteLine(links.Format(objectValueLink));
282             output.WriteLine(links.Format(objectFromGetObject));
283             Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
284         }
285
286         [Fact]
287         public void GetObjectFromObjectValueLinkTest()
288         {
289             ILinks<TLink> links = CreateLinks();
290             var defaultJsonStorage =CreateJsonStorage(links);
291             TLink document = defaultJsonStorage.CreateDocument("documentName");
292             TLink documentObjectValueLink = defaultJsonStorage.AttachObject(document);
293             TLink objectValueLink = links.GetTarget(documentObjectValueLink);
294             TLink objectFromGetObject = defaultJsonStorage.GetObject(objectValueLink);
295             Assert.Equal(links.GetTarget(objectValueLink), objectFromGetObject);
296         }
297
298         [Fact]
299         public void AttachStringValueToKey()
300         {
301             ILinks<TLink> links = CreateLinks();
302             var defaultJsonStorage =CreateJsonStorage(links);
303             TLink document = defaultJsonStorage.CreateDocument("documentName");
304             TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
305             TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
306             TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
307             TLink memberStringValueLink = defaultJsonStorage.AttachString(memberLink,
                ↪  "stringValue");
308             TLink stringValueLink = links.GetTarget(memberStringValueLink);
309             List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
310             Assert.Equal(memberLink, objectMembersLinks[0]);
311             Assert.Equal(stringValueLink,
                ↪  defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
312         }
313
314         [Fact]
315         public void AttachNumberValueToKey()
316         {
317             ILinks<TLink> links = CreateLinks();
318             var defaultJsonStorage =CreateJsonStorage(links);
319             TLink document = defaultJsonStorage.CreateDocument("documentName");
```

```
320             TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
321             TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
322             TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
323             TLink memberNumberValueLink = defaultJsonStorage.AttachNumber(memberLink, 123);
324             TLink numberValueLink = links.GetTarget(memberNumberValueLink);
325             List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
326             Assert.Equal(memberLink, objectMembersLinks[0]);
327             Assert.Equal(numberValueLink,
                ↪  defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
328         }
329
330         [Fact]
331         public void AttachObjectValueToKey()
332         {
333             ILinks<TLink> links = CreateLinks();
334             var defaultJsonStorage =CreateJsonStorage(links);
335             TLink document = defaultJsonStorage.CreateDocument("documentName");
336             TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
337             TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
338             TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
339             TLink memberObjectValueLink = defaultJsonStorage.AttachObject(memberLink);
340             TLink objectValueLink = links.GetTarget(memberObjectValueLink);
341             List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
342             Assert.Equal(memberLink, objectMembersLinks[0]);
343             Assert.Equal(objectValueLink,
                ↪  defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
344         }
345
346         [Fact]
347         public void AttachArrayValueToKey()
348         {
349             ILinks<TLink> links = CreateLinks();
350             var defaultJsonStorage =CreateJsonStorage(links);
351             TLink document = defaultJsonStorage.CreateDocument("documentName");
352             TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
353             TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
354             TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
355             TLink arrayElement = defaultJsonStorage.CreateString("arrayElement");
356             TLink[] array = new TLink[3] { arrayElement, arrayElement, arrayElement };
357             TLink memberArrayValueLink = defaultJsonStorage.AttachArray(memberLink, array);
358             TLink arrayValueLink = links.GetTarget(memberArrayValueLink);
359             List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
360             Assert.Equal(memberLink, objectMembersLinks[0]);
361             Assert.Equal(arrayValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
362         }
363
364         [Fact]
365         public void AttachTrueValueToKey()
366         {
367             ILinks<TLink> links = CreateLinks();
368             var defaultJsonStorage =CreateJsonStorage(links);
369             TLink document = defaultJsonStorage.CreateDocument("documentName");
370             TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
371             TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
372             TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
373             TLink memberTrueValueLink = defaultJsonStorage.AttachBoolean(memberLink, true);
374             TLink trueValueLink = links.GetTarget(memberTrueValueLink);
375             List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
376             Assert.Equal(memberLink, objectMembersLinks[0]);
377             Assert.Equal(trueValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
378         }
379
380         [Fact]
381         public void AttachFalseValueToKey()
382         {
383             ILinks<TLink> links = CreateLinks();
384             var defaultJsonStorage =CreateJsonStorage(links);
385             TLink document = defaultJsonStorage.CreateDocument("documentName");
386             TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
387             TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
388             TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
389             TLink memberFalseValueLink = defaultJsonStorage.AttachBoolean(memberLink, false);
390             TLink falseValueLink = links.GetTarget(memberFalseValueLink);
391             List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
392             Assert.Equal(memberLink, objectMembersLinks[0]);
393             Assert.Equal(falseValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
394         }
395
```

```
396         [Fact]
397         public void AttachNullValueToKey()
398         {
399             ILinks<TLink> links = CreateLinks();
400             var defaultJsonStorage =CreateJsonStorage(links);
401             TLink document = defaultJsonStorage.CreateDocument("documentName");
402             TLink documentObjectValue = defaultJsonStorage.AttachObject(document);
403             TLink @object = defaultJsonStorage.GetObject(documentObjectValue);
404             TLink memberLink = defaultJsonStorage.AttachMemberToObject(@object, "keyName");
405             TLink memberNullValueLink = defaultJsonStorage.AttachNull(memberLink);
406             TLink nullValueLink = links.GetTarget(memberNullValueLink);
407             List<TLink> objectMembersLinks = defaultJsonStorage.GetMembersLinks(@object);
408             Assert.Equal(nullValueLink, defaultJsonStorage.GetValueLink(objectMembersLinks[0]));
409         }
410     }
411 }
```

Index