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 argumentIndex = 0;
            var linksFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "Links file
            ↪  path", args);
            var jsonFilePath = ConsoleHelpers.GetOrReadArgument(argumentIndex++, "JSON file
            ↪  path", args);
            var defaultDocumentName = Path.GetFileNameWithoutExtension(jsonFilePath);
            var documentName = ConsoleHelpers.GetOrReadArgument(argumentIndex, $"Document name
            ↪  (default: {defaultDocumentName})", args);
            if (string.IsNullOrWhiteSpace(documentName))
            {
                documentName = defaultDocumentName;
            }
```

```
            if (!File.Exists(linksFilePath))
            {
                Console.WriteLine($"${linksFilePath} file does not exist.");
            }
            using FileStream jsonFileStream = new(jsonFilePath, FileMode.Append);
            JsonWriterOptions utf8JsonWriterOptions = new()
            {
                Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping,
                Indented = true
            };
            Utf8JsonWriter utf8JsonWriter = new(jsonFileStream, utf8JsonWriterOptions);
            var linksConstants = new LinksConstants<TLink>(enableExternalReferencesSupport:
            ↪  true);
            using UnitedMemoryLinks<TLink> memoryAdapter = new (new
            ↪  FileMappedResizableDirectMemory(linksFilePath),
            ↪  UnitedMemoryLinks<TLink>.DefaultLinksSizeStep, linksConstants,
            ↪  IndexTreeType.Default);
            var links = memoryAdapter.DecorateWithAutomaticUniquenessAndUsagesResolution();
            BalancedVariantConverter<TLink> balancedVariantConverter = new(links);
            var storage = new DefaultJsonStorage<TLink>(links, balancedVariantConverter);
            var exporter = new JsonExporter<TLink>(storage);
            var document = storage.GetDocumentOrDefault(documentName);
            if (storage.EqualityComparer.Equals(document, default))
            {
                Console.WriteLine("No document with this name.");
            }
            using ConsoleCancellation cancellation = new ();
            var cancellationToken = cancellation.Token;
            Console.WriteLine("Press CTRL+C to stop.");
            try
            {
                exporter.Export(document, ref utf8JsonWriter, in cancellationToken);
            }
            catch (Exception exception)
            {
                Console.WriteLine(exception);
                return;
            }
            finally
            {
                utf8JsonWriter.Dispose();
            }
            Console.WriteLine("Export completed successfully.");
        }
    }
}
```

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

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

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

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

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

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

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

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

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

```
    }
}
```

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

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

namespace Platform.Data.Doublets.Json.Tests
{
    public class JsonStorageTests
    {
        private readonly ITestOutputHelper output;
        public static BalancedVariantConverter<TLink> BalancedVariantConverter;

        public JsonStorageTests(ITestOutputHelper output)
        {
            this.output = output;
        }

        public static ILinks<TLink> CreateLinks() => CreateLinks<TLink>(new
        ↪  Platform.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()
        {
            var links = CreateLinks();
            return CreateJsonStorage(links);
        }

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

        [Fact]
        public void ConstructorsTest() => CreateJsonStorage();

        [Fact]
        public void CreateDocumentTest()
        {
            var defaultJsonStorage = CreateJsonStorage();
            defaultJsonStorage.CreateDocument("documentName");
        }

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

        [Fact]
        public void CreateObjectTest()
        {
            var defaultJsonStorage = CreateJsonStorage();
            var object0 = defaultJsonStorage.CreateObjectValue();
            var object1 = defaultJsonStorage.CreateObjectValue();
            Assert.NotEqual(object0, object1);
        }

```

```
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 
220             TLink valueMarker = links.GetSource(createdArrayValue);
221             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
222 
223             TLink createdArrayLink = links.GetTarget(createdArrayValue);
224             TLink arrayMarker = links.GetSource(createdArrayLink);
225             Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
226 
227             TLink createArrayContents = links.GetTarget(createdArrayLink);
228             Assert.Equal(createArrayContents, defaultJsonStorage.EmptyArrayMarker);
```

```
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
                 ↪  => links.GetSource(arrayElementLink) == defaultJsonStorage.ValueMarker);
250             IEnumerable<TLink> arrayElementsValuesLink =
                 ↪  rightSequenceWalker.Walk(createdArrayValue);
251             Assert.NotEmpty(arrayElementsValuesLink);
252
253             output.WriteLine(links.Format(createdArrayValue));
254
255
256             TLink valueMarker = links.GetSource(createdArrayValue);
257             Assert.Equal(valueMarker, defaultJsonStorage.ValueMarker);
258
259             TLink createdArrayLink = links.GetTarget(createdArrayValue);
260             TLink arrayMarker = links.GetSource(createdArrayLink);
261             Assert.Equal(arrayMarker, defaultJsonStorage.ArrayMarker);
262
263             TLink createdArrayContents = links.GetTarget(createdArrayLink);
264             Assert.Equal(links.GetTarget(createdArrayContents), arrayElement);
265
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