JSON-B: Java™ API for JSON Binding

Version 1.0 Early Draft
March 5, 2015

Editors:
Martin Grebac

Comments to: users@jsonb-spec.java.net

Oracle Corporation
500 Oracle Parkway, Redwood Shores, CA 94065 USA.

Contents

Chapter 1

Introduction

This specification defines binding API between Java objects and JSON [1] documents. Readers are assumed to be familiar with JSON; for more information about JSON, see:

- Architectural Styles and the Design of Network-based Software Architectures[2]
- The REST Wiki[3]
- JSON on Wikipedia[4]

1.1 Status

This is an early draft; this specification is not yet complete. A list of open issues can be found at:

http://java.net/jira/browse/JSONB_SPEC

The corresponding Javadocs can be found online at:

http://jsonb-spec.java.net/

The reference implementation will be obtained from:

http://eclipselink.org/

The expert group seeks feedback from the community on any aspect of this specification, please send comments to:

users@jsonb-spec.java.net

1.2 Goals

The following are the goals of the API:

JSON Support binding (marshalling and unmarshalling) for all RFC 7159 compatible JSON documents.

Relationships to JSON Related specifications JSON related specifications will be surveyed to determine their relationship to JSON-Binding.

Consistency Maintain consistency with JAXB (Java API for XML Binding) and other JavaEE and SE APIs.

Convention Define default mapping of Java classes and instances to JSON document counterparts.

Customization Allow to customize the default mapping definition.

Ease Of Use Default use of the APIs SHOULD NOT require prior knowledge of the JSON document format and specification.

Partial Mapping In many usecases, only a subset of JSON Document is required to be mapped to a Java object instance.

Integration Define or enable integration with following Java EE technology standards: JSR 353 - Java API for JSON Processing 1.1 JSR 349, ... - Bean Validation (BV) 1.1 (2.0) JSR 370 - JavaTM API for RESTful Web Services (JAX-RS) 2.1

1.3 Non-Goals

The following are non-goals:

Preserving equivalence (Round-trip) The specification recommends, but does not require equivalence of content for unmarshalled and marshalled JSON documents.

JSON Schema Generation of JSON Schema from Java classes, as well as validation based on JSON schema is out of scope of this specification.

JEP 198 Lightweight JSON API Support Support and integration with Lightweight JSON API as defined within JEP 198 is out of scope of this specification. Will be reconsidered in future specification revisions.

1.4 Conventions

The keywords ‘MUST’, ‘MUST NOT’, ‘REQUIRED’, ‘SHALL’, ‘SHALL NOT’, ‘SHOULD’, ‘SHOULD NOT’, ‘RECOMMENDED’, ‘MAY’, and ‘OPTIONAL’ in this document are to be interpreted as described in RFC 2119[**?**].

Java code and sample data fragments are formatted as shown in figure 1.1:

URIs of the general form ‘http://example.org/...’ and ‘http://example.com/...’ represent application or context-dependent URIs.

All parts of this specification are normative, with the exception of examples, notes and sections explicitly marked as ‘Non-Normative’. Non-normative notes are formatted as shown below.

Note: *This is a note.*

Figure 1.1: Example Java Code

```
package com.example.hello;

public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello World");
    }
}
```

1.5 Terminology

Databinding Process which defines representation of information in an JSON document as an object instance, and vice versa.

Unmarshalling Process of reading a JSON document and construction a tree of content objects, where each object corresponds to part of JSON document, thus the content tree reflects the document's content.

Marshalling Inverse process to unmarshalling. Process of traversing content object tree and writing a JSON document that reflects the tree's content.

1.6 Expert Group Members

This specification is being developed as part of JSR 367 under the Java Community Process. It is the result of the collaborative work of the members of the JSR 367 Expert Group. The following are the present expert group members:

– Martin Grebac (Oracle)
– Martin Vojtek (Oracle)
– Hendrik Saly (Individual Member)
– Gregor Zurowski (Individual Member)
– Inderjeet Singh (Individual Member)
– Eugen Cepoi (Individual Member)
– Przemyslaw Bielicki (Individual Member)
– Kyung Koo Yoon (TmaxSoft, Inc.)
– Otavio Santana (Individual Member)
– Rick Curtis (IBM)
– Alexander Salvanos (Individual Member)

1.7 Acknowledgements

During the course of this JSR we received many excellent suggestions. Special thanks to ... .

During the course of this JSR we received many excellent suggestions on the JSR java.net project mailing lists, thanks in particular to ... for their contributions. The following individuals have also made invaluable technical contributions: ... .

Chapter 2

Runtime API

JSON-B Runtime API provides access to marshalling and unmarshalling operations for manipulating with JSON documents and mapped JSON-B classes and instances. The full specification of the binding framework is available in the javadoc for the `javax.json.bind package`.

Chapter 3

Default Mapping

This section defines the default binding (representation) of Java components and classes within Java programming language to JSON documents. The default binding defined here can be further customized as specified in Chapter 4 - Customizing Mapping.

3.1 General

JSON Binding implementations ('implementations' in further text) MUST support binding of JSON documents as defined in RFC 7159 JSON Grammar. Marshalled JSON output MUST conform to the RFC 7159 JSON Grammar and be encoded in UTF-8 encoding as defined in Section 8.1 (Character Encoding) of RFC 7159. [JSB-3.1-1] Implementations MUST support unmarshalling of documents conforming to RFC 7159 JSON Grammar. [JSB-3.1-2] In addition, implementations SHOULD NOT allow unmarshalling of RFC 7159 non-conforming text (e.g. unsupported encoding, ...) and report error in such case. [JSB-3.1-3] Detection of UTF encoding of unmarshalled document is done as defined in the Section 3 (Encoding) of RFC 4627. [JSB-3.1-4] Implementations SHOULD ignore presence of UTF BOM and not treat it as an error.[JSB-3.1-5]

3.2 Errors

Implementations SHOULD NOT allow unmarshalling of RFC 7159 non-conforming text (e.g. unsupported encoding, ...) and report error in such case. [JSB-3.2-1] Implementation should report error also if the unmarshalled value is not convertable into the expected type. [JSB-3.2-2]

3.3 Basic Java Types

Implementations MUST support binding of the following basic Java classes and their corresponding primitive types: [JSB-3.3-1]

- java.lang.String
- java.lang.Character
- java.lang.Byte

- java.lang.Short
- java.lang.Integer
- java.lang.Long
- java.lang.Float
- java.lang.Double
- java.lang.Boolean

3.3.1 java.lang.String, Character

Instances of type java.lang.String and java.lang.Character are marshalled to JSON String values as defined within RFC 7159 Section 7 (Strings) in UTF-8 encoding without byte order mark. [JSB-3.3.1-1] Implementations SHOULD support unmarshaling of JSON text in other (than UTF-8) UTF encodings into java.lang.String instances.[JSB-3.3.1-2]

3.3.2 java.lang.Byte, Short, Integer, Long, Float, Double

Instances of type java.lang.Byte, Short, Integer, Long, Float, Double and their corresponding primitive types are marshalled to JSON Number with conversion defined in specification for their corresponding toString method [JSB-3.3.2-1]. Unmarshalling of JSON value into java.lang.Byte, Short, Integer, Long, Float, Double instance or corresponding primitive type is done with conversion as defined in the specification for their corresponding parse$Type method, such as java.lang.Byte.parseByte for Byte. [JSB-3.3.2-2]

3.3.3 java.lang.Boolean

Instances of type java.lang.Boolean and its corresponding boolean primitive type are marshalled to JSON value with conversion defined in specification for java.lang.Boolean.toString method [JSB-3.3.3-1]. Unmarshalling of JSON value into java.lang.Boolean instance or boolean primitive type is done with conversion as defined in specification for java.lang.Boolean.parseBoolean method. [JSB-3.3.3-2]

3.3.4 java.lang.Number

Instances of type java.lang.Number (if their more concrete type is not defined elsewhere in this chapter) are marshalled to JSON string by retrieving double value returned from java.lang.Number.doubleValue() method and converting the value to JSON Number as defined in subsection 3.3.2 java.lang.Byte, Short, Integer, Long, Float, Double. [JSB-3.3.4-1].
Unmarshalling of JSON value into Java type java.lang.Number should return instance of java.math.BigDecimal by using conversion as defined in the specification for constructor of java.math.BigDecimal with java.lang.String. [JSB-3.3.4-2]

3.4 Specific Standard Java SE Types

Implementations MUST support binding of the following standard Java SE classes: [JSB-3.4-1]

- java.math.BigInteger

- java.math.BigDecimal
- java.net.URL
- java.net.URI
- java.util.Optional
- java.util.OptionalInt
- java.util.OptionalLong
- java.util.OptionalDouble

3.4.1 java.math.BigInteger, BigDecimal

Instances of type java.math.BigInteger, BigDecimal are marshalled to JSON Number with conversion defined in specification for their toString method [JSB-3.4.1-1]. Unmarshalling of JSON value into java.math.BigInteger, BigDecimal instance is done with conversion as defined in the specification for constructor of java.math.BigInteger, BigDecimal with java.lang.String. [JSB-3.4.1-2]

3.4.2 java.net.URL, URI

Instances of type java.net.URL, URI are marshalled to JSON String value with conversion defined in specification for their toString method [JSB-3.4.2-1]. Unmarshalling of JSON value into java.net.URL, URI instance is done with conversion as defined in the specification for constructor of java.net.URL, URI with java.lang.String input. [JSB-3.4.2-2]

3.4.3 java.util.Optional, OptionalInt, OptionalLong, OptionalDouble

Non-empty instances of type java.util.Optional, OptionalInt, OptionalLong, OptionalDouble are marshalled to JSON value by retrieving their contained instance and converting it to JSON value based on its type and corresponding mapping definitions within this chapter. [JSB-3.4.3-1] Empty optional instances marshalled as object instance properties are ignored in marshalling. [JSB-3.4.3-2] Empty optional instances marshalled as JSON array elements are marshalled as null value [JSB-3.4.3-3]. Unmarshalling into Optional, OptionalInt, OptionalLong, OptionalDouble returns empty optional value for properties which are not present in JSON document or contain null value. [JSB-3.4.2-4] Otherwise any non-empty Optional, OptionalInt, OptionalLong, OptionalDouble value is constructed of type unmarshalled based on mappings defined in this chapter.[JSB-3.4.2-5]

3.5 Enum

Enum instances are marshalled to JSON String value with conversion defined in specification for their toString method [JSB-3.5-1]. Unmarshalling of JSON value into enum instance is done by calling enum's valueOf(String) method. [JSB-3.5-2]

3.6 Interfaces

Implementations MUST support unmarshalling of specific interfaces defined in section 3.7 Collections and subsection 3.3.4 java.lang.Number. [JSB-3.6-1] Unmarshalling to other interfaces is not supported and implementations SHOULD report error in such case. [JSB-3.6-2] If class property is defined with an interface, and not concrete type, mapping for marshalling the property is resolved based on its runtime type.[JSB-3.6-3]

3.7 Collections

Implementations MUST support binding of the following collection interfaces, classes and their implementations. [JSB-3.7-1] Implementations of interfaces below MUST provide accessible default constructor.[JSB-3.7-2] JSON Binding implementations MUST report unmarshalling error if default constructor is not present or is not in accessible scope.[JSB-3.7-3]

- java.util.Collection
- java.util.Map
- java.util.Set
- java.util.HashSet
- java.util.NavigableSet
- java.util.SortedSet
- java.util.TreeSet
- java.util.LinkedHashSet
- java.util.TreeHashSet
- java.util.HashMap
- java.util.NavigableMap
- java.util.SortedMap
- java.util.TreeMap
- java.util.LinkedHashMap
- java.util.TreeHashMap
- java.util.List
- java.util.ArrayList
- java.util.LinkedList
- java.util.Deque
- java.util.ArrayDeque
- java.util.Queue
- java.util.PriorityQueue
- java.util.EnumSet
- java.util.EnumMap

For interfaces defined above, following table defines default implementation types. Default implementation type for a class field or property with interface defined type is used as concrete runtime type to unmarshall JSON values into this property. [JSB-3.7-4]

Interface	Default implementation type
java.util.Collection	java.util.ArrayList
java.util.Set	java.util.HashSet
java.util.NavigableSet	java.util.TreeSet
java.util.SortedSet	java.util.TreeSet
java.util.Map	java.util.HashMap
java.util.SortedMap	java.util.TreeMap
java.util.NavigableMap	java.util.TreeMap
java.util.Deque	java.util.ArrayDeque
java.util.Queue	java.util.ArrayDeque

3.8 Names and identifiers

3.9 Java Package

3.10 Java Class

Chapter 4

Customizing Mapping

JSON-B TBD

Bibliography

[1] Ed. T. Bray. The javascript object notation (json) data interchange format. RFC 2070-1721, IETF, March 2014.

[2] R. Fielding. Architectural Styles and the Design of Network-based Software Architectures. Ph.d dissertation, University of California, Irvine, 2000. See http://roy.gbiv.com/pubs/dissertation/top.htm.

[3] REST Wiki. Web site. See http://rest.blueoxen.net/cgi-bin/wiki.pl.

[4] Technical report, Wikipedia, November 2014.