JSONB introduction and comparison with other frameworks

Dmitry Kornilov
JSONB spec lead

dmitry.kornilov@oracle.com

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

1. What is JSON-B
2. What is in the spec
3. Default mapping
4. Customized mapping
5. Q&A

What is JSON Binding?

- JSON Binding is a standard
- It’s about converting Java objects to and from JSON documents
- JSON Binding = JSON-B = JSONB = JSR 367

What is JSON Binding?

Java

```
public class Customer {
    public int id;
    public String firstName;
    public String lastName;
   ....
}
```

```
Customer e = new Customer();
e.id = 1;
e.firstName = “John”;
e.lastName = “Doe”;
```

JSON

```
{
   "id": 1,
   "firstName" : "John",
   "lastName" : "Doe",
}
```

What is JSON Binding?

Java

```
public class Customer {
    public int id;
    public String firstName;
    public String lastName;
   ....
}
```

```
Customer e = new Customer();
e.id = 1;
e.firstName = “John”;
e.lastName = “Doe”;
```

JSON-B

JSON

```
{
   "id": 1,
   "firstName" : "John",
   "lastName" : "Doe",
}
```

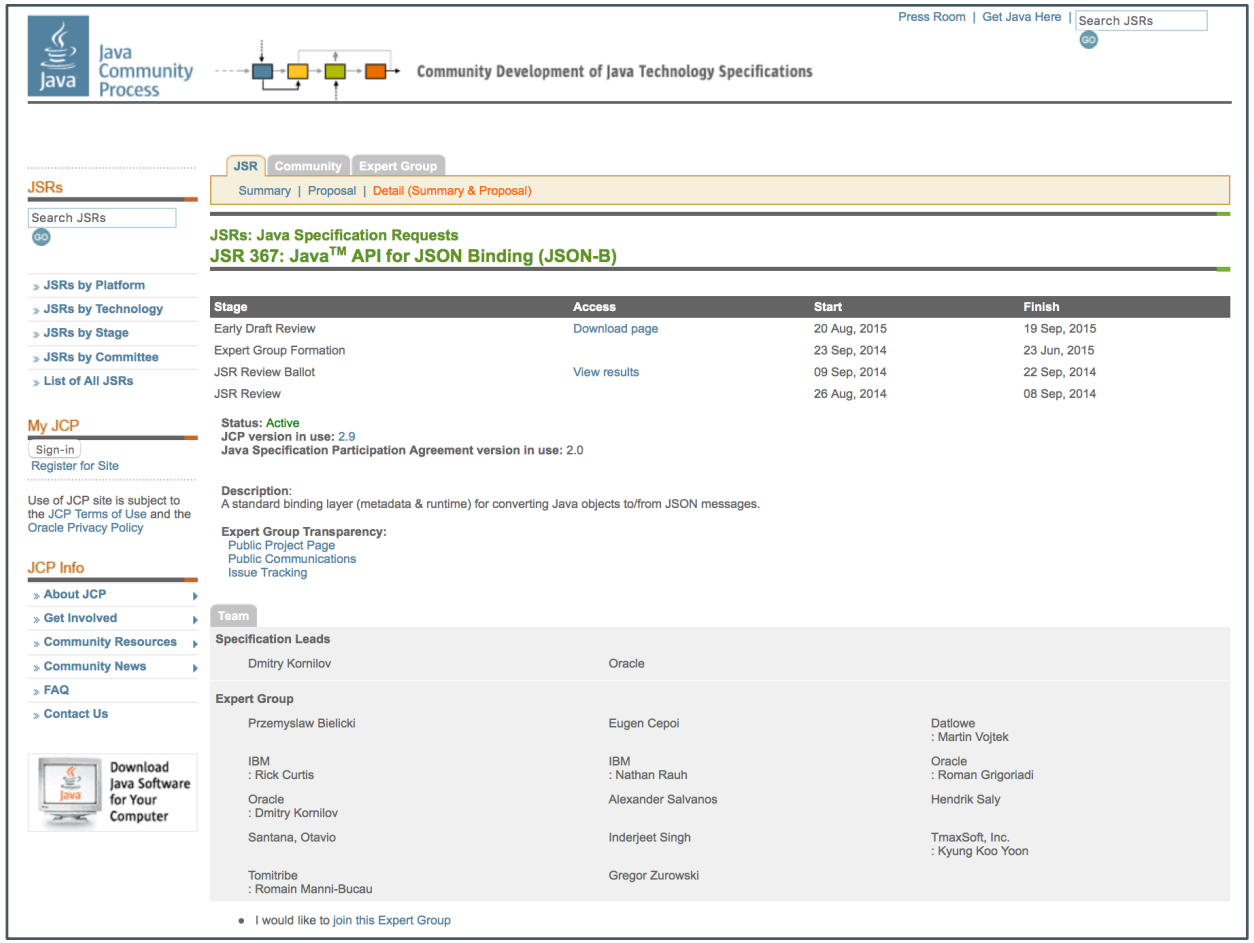
Alternatives

- Genson
- Gson
- Jackson
- …

JSON-B Specification

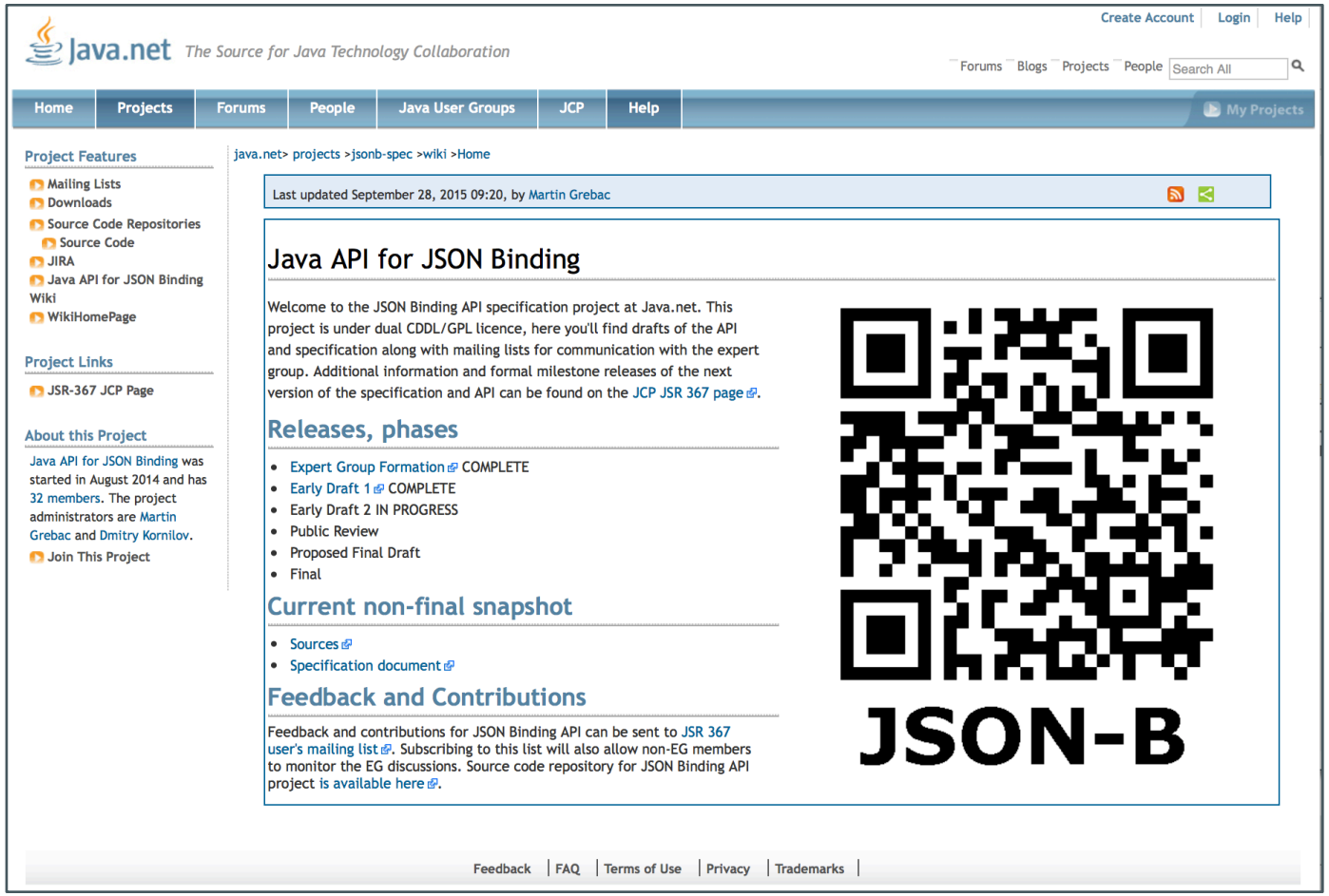
Java Community Process

- JSR-367
- JSR status and updates
- Expert group



Specification and API Project

- Hosted on java.net
- Spec in pdf format
- Git repository
- Wiki
- Bug tracker
- Mailing lists

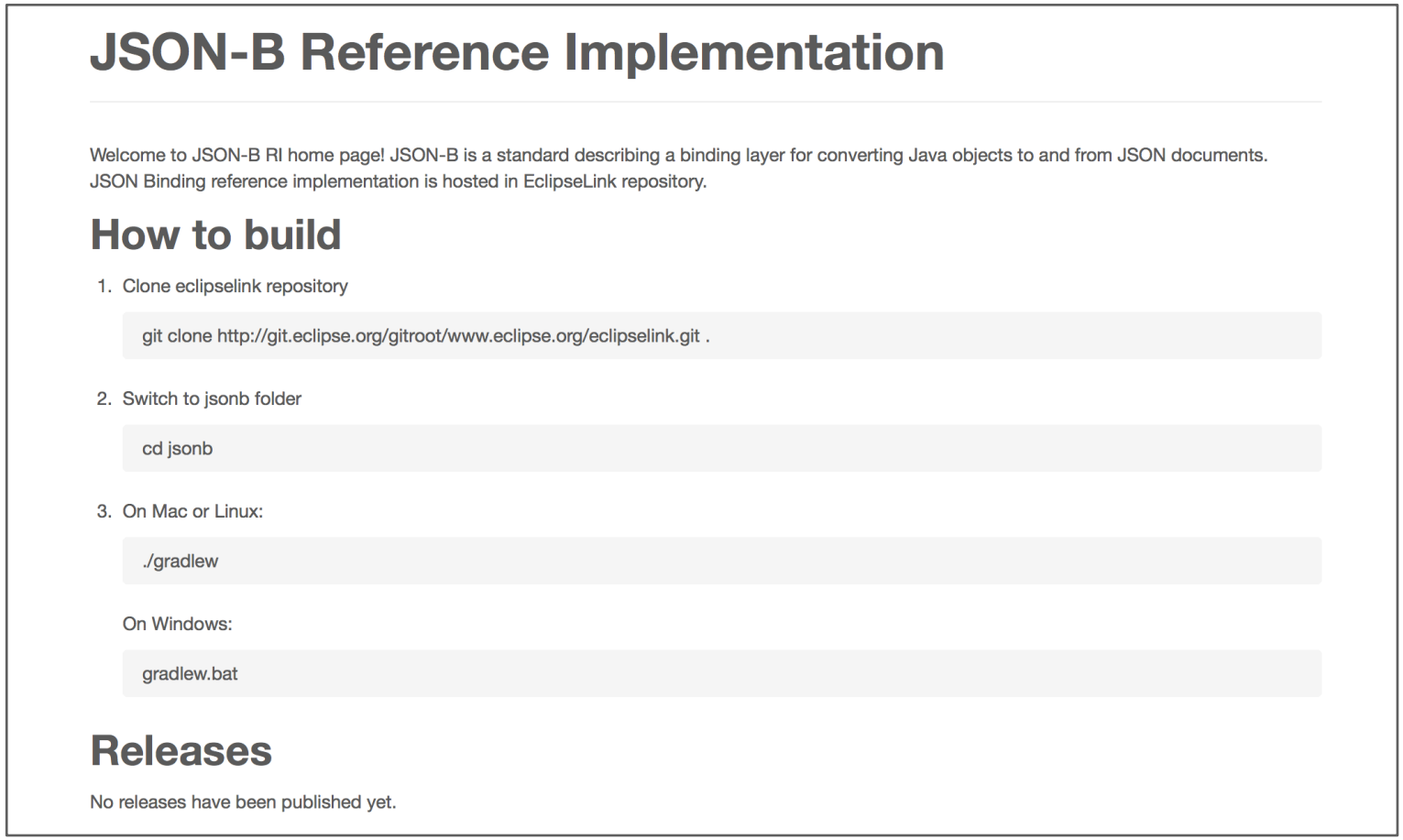


Participate!

users@jsonb-spec.java.net

Reference Implementation

- eclipselink.org/jsonb
- Mirror on GitHub



JSON-B Reference Implementation

Welcome to JSON-B RI home page! JSON-B is a standard describing a binding layer for converting Java objects to and from JSON documents. JSON Binding reference implementation is hosted in EclipseLink repository.

How to build

1. Clone eclipselink repository

```
git clone http://git.eclipse.org/gitroot/www.eclipse.org/eclipselink.git .
```

2. Switch to jsonb folder

```
cd jsonb
```

3. On Mac or Linux:

```
./gradlew
```

On Windows:

```
gradlew.bat
```

Releases

No releases have been published yet.

Summary

- JCP Page
  https://www.jcp.org/en/jsr/detail?id=367
- Specification Project Home:
  https://java.net/projects/jsonb-spec
- API sources & samples:
  https://java.net/projects/jsonb-spec/sources/git/show/api
- Specification in pdf:
  https://java.net/projects/jsonb-spec/sources/git/content/spec/spec.pdf
- Reference implementation:
  http://eclipselink.org/jsonb

JSON-B Default Mapping

Default Mapping

- No configuration, no annotations
- The scope:
  - Basic Types
  - Specific JDK Types
  - Dates
  - Classes
  - Collections/Arrays
  - Enumerations
  - JSON-P

```
import javax.json.bind.Jsonb;
import javax.json.bind.JsonbBuilder;

// Create with default config
Jsonb jsonb = JsonbBuilder.create();
```

JSON-B Engine

- toJson(…)
- fromJson(…)

```
String toJson(Object object);
String toJson(Object object, Type runtimeType);
void toJson(Object object, Writer writer);
void toJson(Object object, Type runtimeType, Writer appendable);
void toJson(Object object, OutputStream stream);
void toJson(Object object, Type runtimeType, OutputStream stream);


<T> T fromJson(String str, Class<T> type);
<T> T fromJson(String str, Type runtimeType);
<T> T fromJson(Reader readable, Class<T> type);
<T> T fromJson(Reader readable, Type runtimeType);
<T> T fromJson(InputStream stream, Class<T> type);
<T> T fromJson(InputStream stream, Type runtimeType);
```

Default Mapping – Basic Types

- java.lang.String
- java.lang.Character
- java.lang.Byte (byte)
- java.lang.Short (short)
- java.lang.Integer (int)
- java.lang.Long (long)
- java.lang.Float (float)
- java.lang.Double (double)
- java.lang.Boolean (boolean)

Default Mapping – Basic Types

- java.lang.String
- java.lang.Character
- java.lang.Byte (byte)
- java.lang.Short (short)
- java.lang.Integer (int)
- java.lang.Long (long)
- java.lang.Float (float)
- java.lang.Double (double)
- java.lang.Boolean (boolean)

Serialization:
toString()

Deserialization:
parseXXX()

Default Mapping – Basic Types

• java.lang.String	**“string”**	**“string”**
• java.lang.Character	**’\u0041’**	**“A”**
• java.lang.Byte (byte)	(**byte**) 1	1
• java.lang.Short (short)	(**short**) 1	1
• java.lang.Integer (int)	(**int**) 1	1
• java.lang.Long (long)	1L	1
• java.lang.Float (float)	1.2f	1.2
• java.lang.Double (double)	1.2	1.2
• java.lang.Boolean (boolean)	**true**	**true**

Default Mapping – Specific Types

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

Default Mapping – Specific Types

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

Serialization:
toString()

Deserialization:
Single argument constructor

Default Mapping – Specific Types

- java.math.BigInteger
- java.math.BigDecimal
- java.net.URL
- java.net.URI

Serialization:
toString()

Deserialization:
Single argument constructor

- java.util.Optional
- java.util.OptionalInt
- java.util.OptionalLong
- java.util.OptionalDouble

- Represented by its value if not empty
- Considered null if empty

Optional Types

```
OptionalInt.of(1)
```

JSON-B	1
Genson	{"asInt": 1, "present": true}
Gson	{"value": 1, "present": true}
Jackson	1

```
OptionalInt.empty()
```

JSON-B	null
Genson	–
Gson	{"value": 0, "present": false}
Jackson	null

Default Mapping – Dates

java.util.Date	ISO_DATE_TIME
java.util.Calendar, java.util.GregorianCalendar	ISO_DATE if to time information present, otherwise ISO_DATE_TIME
Java.util.TimeZone, java.util.SimpleTimeZone	NormalizedCustomId (see TimeZone javadoc)
java.time.Instant	ISO_INSTANT
java.time.LocalDate	ISO_LOCAL_DATE
java.time.LocalTime	ISO_LOCAL_TIME
java.time.LocalDateTime	ISO_LOCAL_DATE_TIME
java.time.ZonedDateTime	ISO_ZONED_DATE_TIME
java.time.OffsetDateTime	ISO_OFFSET_DATE_TIME
java.time.OffsetTime	ISO_OFFSET_TIME
java.time.ZoneId	NormalizedZoneId as specified in ZoneId javadoc
java.time.ZoneOffset	NormalizedZoneId as specified in ZoneOffset javadoc
java.time.Duration	ISO 8601 seconds based representation
java.time.Period	ISO 8601 period representation

Default Mapping – Date Sample

```
SimpleDateFormat sdf =
  new SimpleDateFormat("dd.MM.yyyy");
Date date = sdf.parse("08.03.2016");
```

JSON-B	“2016-03-08T00:00:00”
Genson	1457391600000
Gson	"Mar 8, 2016 12:00:00 AM”
Jackson	1457391600000

Default Mapping – Calendar Sample

```
Calendar cal = Calendar.getInstance();
cal.clear();
cal.set(2016, 3, 8);
```

JSON-B	“2016-03-08”
Genson	1457391600000
Gson	{"year":2016, "month":3, "dayOfMonth":8, "hourOfDay":0, "minute":0, "second": 0}
Jackson	1457391600000

Default Mapping – Classes

- Public and protected **nested** and **static nested** classes
- **Anonymous** classes (serialization only)
- Inheritance is supported
- Default no-argument constructor is required for deserialization

Default Mapping – Fields

- **Final** fields are serialized
- **Static** fields are skipped
- **Transient** fields are skipped
- **Null** fields are skipped
- **Lexicographical** order
- Parent class fields are serialized before child class fields

Fields Order Comparison

```
class Parent {
    public int parentB = 2;
    public int parentA = 1;
}

class Child extends Parent {
    public int childB = 4;
    public int childA = 3;
}
```

JSON-B	{"parentA":1,"parentB":2, "childA":3, "childB":4}
Genson	{"childA":3, "childB":4, "parentA":1,"parentB":2}
Gson	{"childB":4, "childA":3, "parentB":2,"parentA":1}
Jackson	{"parentB":2,"parentA":1, "childB":4, "childA":3}

Default Mapping – Scope and Field Access Strategy

Serialization

- Existing fields with public getters
- Public fields with no getters
- Public getter/setter pair without a corresponding field

Deserialization

- Existing fields with public setters
- Public fields with no setters
- Public getter/setter pair without a corresponding field

Scope and Field Access Strategy – JSON-B

```
public class Foo {
    public final int publicFinalField;
    private final int privateFinalField;

    public static int publicStaticField;

    public int publicWithNoGetter;
    public int publicWithPrivateGetter;
    public Integer publicNullField = null;

    private int privateWithNoGetter;
    private int privateWithPublicGetter;

    public int getNoField() {};
    public void setNoField(int value) {};
}
```

```
{
  "publicFinalField": 1,

  "publicWithNoGetter": 1,

  "privateWithPublicGetter": 1,

  "noField": 1
}
```

Scope and Field Access Strategy – Genson

```java
public class Foo {
    public final int publicFinalField;
    private final int privateFinalField;

    public static int publicStaticField;

    public int publicWithNoGetter;
    public int publicWithPrivateGetter;
    public Integer publicNullField = null;

    private int privateWithNoGetter;
    private int privateWithPublicGetter;

    public int getNoField() {};
    public void setNoField(int value) {};
}
```

```json
{
  "publicFinalField": 1,

  "publicWithNoGetter": 1,
  "publicWithPrivateGetter": 1,
  "publicNullField": null,

  "privateWithPublicGetter": 1,

  "noField": 1
}
```

Scope and Field Access Strategy – Gson

```
public class Foo {
    public final int publicFinalField;
    private final int privateFinalField;

    public static int publicStaticField;

    public int publicWithNoGetter;
    public int publicWithPrivateGetter;
    public Integer publicNullField = null;

    private int privateWithNoGetter;
    private int privateWithPublicGetter;

    public int getNoField() {};
    public void setNoField(int value) {};
}
```

```
{
  "publicFinalField": 1,
  "privateFinalField": 1,

  "publicWithNoGetter": 1,
  "publicWithPrivateGetter": 1,

  "privateWithNoGetter": 1,
  "privateWithPublicGetter": 1,
}
```

Scope and Field Access Strategy – Jackson

```
public class Foo {
    public final int publicFinalField;
    private final int privateFinalField;

    public static int publicStaticField;

    public int publicWithNoGetter;
    public int publicWithPrivateGetter;
    public Integer publicNullField = null;

    private int privateWithNoGetter;
    private int privateWithPublicGetter;

    public int getNoField() {};
    public void setNoField(int value) {};
}
```

```
{
  "publicFinalField": 1,

  "publicWithNoGetter": 1,
  "publicWithPrivateGetter": 1,
  "publicNullField": null,

  "privateWithPublicGetter": 1,

  "noField": 1
}
```

Scope and Field Access Strategy – Summary

Framework	Respects getters/setters	Strict getter/setter	Private fields	Virtual fields
JSON-B	Yes	Yes	No	Yes
Genson	Yes	No	No	Yes
Gson	No	No	Yes	No
Jackson	Yes	No	No	Yes

Arrays/Collections

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

```
// Array
int[] intArray = {1, 2, 3};

jsonb.toJson(intArray);       // [1,2,3]

// Collection
Collection<Object> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(null);

jsonb.toJson(list);           // [1,2,null]

// Map
Map<String, Object> map = new LinkedHashMap<>();
map.put("first", 1);
map.put("second", 2);

jsonb.toJson(map);            // {"first":1,"second":2}
```

JSON-P Types

- javax.json.JsonArray
- javax.json.JsonStructure
- javax.json.JsonValue
- javax.json.JsonPointer
- javax.json.JsonString
- javax.json.JsonNumber

JSON-P Types

- javax.json.JsonArray
- javax.json.JsonStructure
- javax.json.JsonValue
- javax.json.JsonPointer
- javax.json.JsonString
- javax.json.JsonNumber

Serialization:
javax.json.JsonWriter

Deserialization:
javax.json.JsonReader

JSON-P Sample

```
JsonBuilderFactory f =
    Json.createBuilderFactory(null);

JsonObject jsonObject = f.createObjectBuilder()
        .add("name", "Jason")
        .add("city", "Prague")
        .build();
```

```
{
   "name": "Jason",
   "city": "Prague"
}
```

JSON-P Types Support in Other Frameworks

- Genson:
  - Support added in JSR353Bundle
- Gson
  - No JSON-P support
- Jackson
  - Support added in JSR353Module

Customized Mapping

JSON-B Engine Configuration

- Annotations
- Runtime configuration
  - JsonbConfig
  - JsonbBuilder

```
JsonbConfig config = new JsonbConfig()
    .withFormatting(…)
    .withNullValues(…)
    .withEncoding(…)
    .withStrictIJSON(…)
    .withPropertyNamingStrategy(…)
    .withPropertyOrderStrategy(…)
    .withPropertyVisibilityStrategy(…)
    .withAdapters(…)
    .withBinaryDataStrategy(…);

Jsonb jsonb = JsonbBuilder.newBuilder()
        .withConfig(…)
        .withProvider(…)
        .build();
```

Customizations

- Property names
- Property order
- Ignoring properties
- Null handling
- Custom instantiation
- Fields visibility
- Adapters
- Date/Number Formats
- Binary Encoding

Property Name Customization

- JSON-B:
  - @JsonbProperty (Field, Method)
- Genson:
  - @JsonProperty (Field, Method)
  - Use GensonBuilder().rename() method
- Gson:
  - @SerializedName (Field)
- Jackson:
  - @JsonProperty (Field, Method)

Custom Mapping - Property Names

```
public class Customer {
    public int id;
    public String firstName;
}
```

```
{
  "id": 1,
  "firstName": "Jason"
}
```

Custom Mapping - Property Names

```
public class Customer {
    private int id;

    @JsonbProperty("name")
    private String firstName;
}
```

```
public class Customer {
    public int id;
    public String firstName;

    @JsonbProperty("name")
    public String getFirstName() {
        return firstName;
    }
}
```

```
{
  "id": 1,
  "name": "Jason"
}
```

Custom Mapping - Property Names

```
public class Customer {
    public int id;
    public String firstName;

    @JsonbProperty(“getter-name")
    String getFirstName() {
        return firstName;
    }

    @JsonbProperty(“setter-name")
    void setFirstName(String str) {
        this.firstName = str;
    }
}
```

Serialization:

```
{
  "id": 1,
  “getter-name": "Jason"
}
```

Deserialization:

```
{
  "id": 1,
  “setter-name": "Jason"
}
```

Property Naming Strategy

- Supported naming strategies
  - IDENTITY (myMixedCaseProperty)
  - LOWER_CASE_WITH_DASHES (my-mixed-case-property)
  - LOWER_CASE_WITH_UNDERSCORES (my_mixed_case_property)
  - UPPER_CAMEL_CASE (MyMixedCaseProperty)
  - UPPER_CAMEL_CASE_WITH_SPACES (My Mixed Case Property)
  - CASE_INSENSITIVE (mYmIxEdCaSePrOpErTy)
  - Or a custom implementation
- JsonbConfig().withPropertyNamingStrategy(…):

Property Naming Strategy

- Genson
  - `GensonBuilder.with(PropertyNameResolver… resolvers)`
- Gson:
  - `GsonBuilder.setFieldNamingPolicy(FieldNamingPolicy policy)`
- Jackson
  - `ObjectMapper.setPropertyNamingStrategy(PropertyNamingStrategy pns)`

Property Order Strategy

- Strategies:
  - LEXICOGRAPHICAL (A-Z)
  - ANY
  - REVERSE (Z-A)
- Compile Time:
  - @JsonbPropertyOrder on class
- Runtime:
  - withPropertyOrderStrategy(…)

```
@JsonbPropertyOrder(ANY)
public class Foo {
    public int bar2;
    public int bar1;
}
```

Ignoring Properties and Visibility Customization

- Compile Time:
  - Transient modifier
  - @JsonbTransient annotation
  - PropertyVisibilityStrategy interface
  - @JsonbVisibility annotation
- Runtime:
  - withPropertyVisibilityStrategy(…)

```
public class Foo {
    public transient int skipped;

    @JsonbTransient
    public int alsoSkipped;
}

@JsonbVisibility(MyStrategy.class)
public class Bar {
    private int field1;
    private int field2;
}
```

Ignoring Properties in Other Frameworks

- Genson
  - @JsonIgnore annotation
  - Include(…) and exclude(…) methods in GensonBuilder class
- Gson:
  - @Exposed annotation
  - ExclusionStrategy interface
- Jackson
  - @JsonIgnore annotation on field
  - @JsonIgnoreProperties annotation on class
  - Filters and mix-ins

Null Handling

- Null fields are skipped by default
- Compile Time:
  - @JsonbNillable annotation
- Runtime:
  - JsonbConfig().withNullValues(true)

```
public class Customer {
    public int id = 1;

    @JsonbNillable
    public String name = null;
}
```

Null Handling in Other Frameworks

Framework	Serializes nulls by default	Null Handling
JSON-B	No	@JsonbNillable
Genson	Yes	GensonBuilder.setSkipNull(true)
Gson	No	GsonBuilder.serializeNulls()
Jackson	Yes	@JsonInclude(JsonInclude.Include.NON_NULL) ObjectMapper.setSerializationInclusion(JsonInclude.Include.*NON_NULL*);

Custom Instantiation

```
public class Customer {
    public int id;
    public String name;

    @JsonbCreator
    public static Customer getFromDb(int id) {
        return CustomerDao.getByPrimaryKey(id);
    }
}

public class Order {
    public int id;
    public Customer customer;
}
```

```
{
  "id": 1,
  "customer": {
    "id": 2  }
}
```

Adapters

- JAXB inspired JsonbAdapter interface
- @JsonbTypeAdapter annotation
- JsonbConfig().withAdapters(JsonbAdapter… adapters);

Adapters Sample

```
public class Car {
    public Integer distance; // In Miles
}
```

Adapters Sample

```
public class Car {
    public Integer distance; // In Miles
}

public class AdaptedCar {
    public Integer distance; // In Kilometers
}
```

Adapters Sample

```
public class Car {
    public Integer distance; // In Miles
}

public class AdaptedCar {
    public Integer distance; // In Kilometers
}

public class CarAdapter implements JsonbAdapter<Car, AdaptedCar> {
    public AdaptedCar toJson(Car car) {
        AdaptedCar adaptedCar = new AdaptedCar();
        adaptedCar.distance = car.distance * 1,60934;
        return adaptedCar;
    }

    public Car fromJson(AdaptedCar adaptedCar) {
        Car car = new Car();
        car.distance = adaptedCar.distance / 1,60934;
        return car;
    }
}
```

Adapters Sample

```
JsonbConfig config = new JsonbConfig()
    .withAdapters(new CarAdapter());
Jsonb jsonb = JsonbBuilder.create(config);

Car car = new Car();
car.distance = 100; // miles

String json = jsonb.toJson(car);
```

```
{
  "distance": 160
}
```

Adapters in Other Frameworks

- Genson
  - Converter interface
  - JAXB adapters support with JAXBBundle
- Gson:
  - TypeAdapter interface
- Jackson
  - External serializers
  - Mix-In annotations

Date/Number Format

```
public class FormatTest {
    public Date defaultDate;

    @JsonbDateFormat("dd.MM.yyyy")
    public Date formattedDate;

    public BigDecimal defaultNumber;

    @JsonbNumberFormat(“#0.00")
    public BigDecimal formattedNumber;
}
```

```
{
  “defaultDate”: “2015-07-26T23:00:00",

  “formattedDate”: ”26.07.2015",

  “defaultNumber": 1.2,

  “formattedNumber": 1.20
}
```

Date/Number Formats in other Frameworks

- Genson
  - @JsonDateFormat annotation
  - GensonBuilder.setDateFormat(dateFormat)
- Gson:
  - GsonBuilder.setDateFormat
- Jackson
  - @JsonFormat annotation
  - objectMapper.setDateFormat(myDateFormat);

Binary Data Encoding

- BYTE (default)
- BASE_64
- BASE_64_URL

```
JsonbConfig config = new JsonbConfig()
   .withBinaryDataStrategy(BinaryDataStrategy.BASE_64);

Jsonb jsonb = JsonbBuilder.create(config);
String json = jsonb.toJson(obj);
```

I-JSON

- I-JSON (”Internet JSON”) is a restricted profile of JSON
  - https://tools.ietf.org/html/draft-ietf-json-i-json-06
- JSON-B fully supports I-JSON by default with three exceptions:
  - JSON Binding does not restrict the serialization of top-level JSON texts that are neither objects nor arrays. The restriction should happen at application level.
  - JSON Binding does not serialize binary data with base64url encoding.
  - JSON Binding does not enforce additional restrictions on dates/times/duration.
- Configuration: withStrictIJSONSerializationCompliance

Special Thanks

- Roman Grigoriadi and David Kral from JSON-B team
- All members of JSON-B experts group
- others on users@jsonb-spec.java.net

Q&A


ORACLE®