Holon
PLATFORM

Holon JSON support

Version 5.0.4

Table of Contents

1. Introduction . . . 1
2. Obtaining the artifacts . . . 1
   2.1. Using the Platform BOM . . . 2
3. Dealing with properties and `PropertySet` . . . 2
   3.1. Serialization . . . 2
   3.2. Deserialization . . . 2
4. Gson integration . . . 3
   4.1. Configuration and use . . . 3
   4.2. JAX-RS integration . . . 4
      4.2.1. `PropertyBox` deserialization . . . 5
      4.2.2. JAX-RS integration configuration . . . 7
   4.3. Spring integration . . . 7
   4.4. Spring Boot integration . . . 7
5. Jackson integration . . . 8
   5.1. Configuration and use . . . 8
   5.2. JAX-RS integration . . . 9
      5.2.1. `PropertyBox` deserialization . . . 10
      5.2.2. JAX-RS integration configuration . . . 12
   5.3. Spring integration . . . 12
   5.4. Spring Boot integration . . . 12
6. Loggers . . . 13
7. System requirements . . . 13
   7.1. Java . . . 13

Copyright © 2016-2017

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

1. Introduction

The Holon Platform **JSON** module provides integration support between the platform foundation PropertyBox data structure and the most popular *JSON* processing libraries:

- Gson
- Jackson

The module faces the following integration concerns:

- *Gson* **GsonBuilder** configuration
- *Jackson* **ObjectMapper** configuration
- **JAX-RS** integration and auto-configuration
- *Spring* **RestTemplate** configuration
- **Spring boot** auto-configuration

2. Obtaining the artifacts

The Holon Platform uses Maven for projects build and configuration. All the platform artifacts are published in the **Maven Central Repository**, so there is no need to explicitly declare additional repositories in your project `pom` file.

At the top of each *section* of this documentation you will find the Maven *coordinates* (group id, artifact id and version) to obtain the artifact(s) as a dependency for your project.

A **BOM (Bill Of Materials)** `pom` is provided to import the available dependencies for a specific version in your projects. The Maven coordinates for the core BOM are the following:

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-json-bom</artifactId>
<version>5.0.4</version>
```

The BOM can be imported in a Maven project in the following way:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.holon-platform.json</groupId>
      <artifactId>holon-json-bom</artifactId>
      <version>5.0.4</version>
      <strong><type>pom</type></strong>
      <strong><scope>import</scope></strong>
    </dependency>
  </dependencies>
</dependencyManagement>
```

2.1. Using the Platform BOM

The Holon Platform provides an **overall Maven BOM (Bill of Materials)** to easily obtain all the available platform artifacts.

See Obtain the platform artifacts for details.

3. Dealing with properties and `PropertySet`

A PropertyBox is a versatile and general-purpose data container object which uses Property type keys as reference for the data it manages.

The **Properties** to which data are bound, i.e. the `PropertySet` associated with the `PropertyBox`, are complex and extensible objects, highly customizable by the developer. For this reason, a `Property` is **never serialized in JSON** by this module.

3.1. Serialization

Only `Path` properties are serialized into *JSON*, using the path **name** as property name of the produced *JSON object* and *JSON* data type consistent with the property type.

This way, the *JSON* definitions produced by the serialization of a `PropertyBox` are fully analogous to a standard *JSON* **object**, composed by a list of property *names* associated to a *value* of a suitable type, which can be eventually deserialized in a generic data object (with a consistent internal structure) even a language different from Java.

3.2. Deserialization

To deserialize a *JSON object* into a `PropertyBox` instance, the `PropertySet` to use for the `PropertyBox` is required and must be available as a Context resource (typically thread-bound).

To bound a `PropertySet` instance to the default thread-scoped resource set of the Holon Platform *Context*, the `execute(Callable operation)` method can be used.

See the next sections for further details and use-case examples.

4. Gson integration

A `JsonSerializer` and a `JsonDeserializer` are provided to configure PropertyBox serialization to *JSON* format and deserialization from *JSON* format.

4.1. Configuration and use

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-gson</artifactId>
<version>5.0.4</version>
```

To enable `PropertyBox` handling in Gson, registering the suitable `JsonSerializer` and a `JsonDeserializer` pair, the GsonConfiguration utility class can be used:

```
GsonBuilder builder = GsonConfiguration.builder(); ①
Gson gson = builder.create();

GsonBuilder mybuilder = getGsonBuilder(); ②
GsonConfiguration.configure(builder); ③
gson = mybuilder.create();
```

① Get a pre-configured `GsonBuilder` with `PropertyBox` support

② Get a previously available `GsonBuilder`

③ Configure the builder for `PropertyBox` support

With a properly configured `Gson` instance, a `PropertyBox` can be serialized and deserialized just like any another object. As described in Dealing with properties and `PropertySet`, you need to declare the `PropertySet` to use as a *Context* resource at `PropertyBox` deserialization time.

```
final static PathProperty<Long> ID = PathProperty.create("id", Long.class);
final static PathProperty<String> DESCRIPTION = PathProperty.create("description",
String.class);

final static PropertySet<?> PROPERTY_SET = PropertySet.of(ID, DESCRIPTION);

public void serializeAndDeserialize() {
  Gson gson = GsonConfiguration.builder().create(); ①

  PropertyBox box = PropertyBox.builder(PROPERTY_SET).set(ID, 1L).set(DESCRIPTION,
"Test").build(); ②

  // serialize
  String json = gson.toJson(box); ③
  // deserialize
  box = PROPERTY_SET.execute(() -> gson.fromJson(json, PropertyBox.class)); ④
}
```

① Obtain a pre-configured `Gson` instance

② Build a `PropertyBox` using `PROPERTY_SET` as property set

③ Serialize the `PropertyBox` to JSON.

④ Deserialize back the JSON definition to a `PropertyBox` instance using `PROPERTY_SET` as property set, declaring it as thread-bound *Context* resource through the `execute(…)` method

In the example above, the `PropertyBox` instance will be serialized as a JSON object like this:

```
{
  "id": 1,
  "description": "Test"
}
```

4.2. JAX-RS integration

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-gson-jaxrs</artifactId>
<version>5.0.4</version>
```

A set of JAX-RS extension classes are provided to configure `PropertyBox` JSON support in JAX-RS using *Gson*, and in a more general sense, to configure *Gson* as main serializer/deserializer in a JAX-RS server and/or client for the JSON media type.

The JAX-RS *Gson* extensions relies on standard `ContextResolver` type to provide the `Gson` instance to use to perform JSON serialization and deserialization. This `Gson` instance must be configured with

`PropertyBox` support as described above to handle `PropertyBox` types consistently.

To setup *Gson* as provider for JSON message type handling, the GsonFeature JAX-RS `Feature` can be registered in the JAX-RS application.

If you use Jersey or Resteasy as JAX-RS implementation, there is no need to explicitly register the `GsonFeature`, just ensure the `holon-gson` jar is in classpath and the *Gson* support will be configured automatically, leveraging Jersey *AutoDiscoverable* and Resteasy Java Service extensions features.

4.2.1. `PropertyBox` deserialization

When a `PropertyBox` is used as a JAX-RS resource method **parameter** (for methods which declare to consume `application/json` media type), the JSON deserialization of the input into a `PropertyBox` instance needs to know the `PropertySet` to use in order to create the property box. For this purpose, the `@PropertySetRef` annotation can be used at method parameter level to declare the `PropertySet` instance to use to deserialize the property box.

The PropertySetRef annotation allows to declare the `PropertySet` instance as the `public static` **field** of a given class, which must be specified in the `value()` annotation attribute. If more than one `public static` field of `PropertySet` type is present in declared class, the `field()` annotation attribute can be used to specify the right field name.

```
final static PathProperty<Integer> CODE = PathProperty.create("code", Integer.class);
final static PathProperty<String> NAME = PathProperty.create("name", String.class);

final static PropertySet<?> PROPERTYSET = PropertySet.of(CODE, NAME);

// JAX-RS example endpoint
@Path("test")
public static class Endpoint {

  @PUT
  @Path("serialize")
  @Consumes(MediaType.APPLICATION_JSON)
  public Response create(@PropertySetRef(value = ExampleGson.class, field =
"PROPERTYSET") PropertyBox data) {
    return Response.accepted().build();
  }

  @GET
  @Path("deserialize")
  @Produces(MediaType.APPLICATION_JSON)
  public PropertyBox getData() {
    return PropertyBox.builder(PROPERTYSET).set(CODE, 1).set(NAME, "Test").build();
  }

}

public void jaxrs() {
  Client client = ClientBuilder.newClient(); ①

  PropertyBox box1 = PropertyBox.builder(PROPERTYSET).set(CODE, 1).set(NAME, "Test")
.build();

  client.target("https://host/test/serialize").request().put(Entity.entity(box1,
MediaType.APPLICATION_JSON)); ②

  PropertyBox box2 = PROPERTYSET
      .execute(() -> client.target("https://host/test/deserialize").request().get
(PropertyBox.class)); ③

}
```

① Create a JAX-RS `Client`

② Perform a `PUT` request providing a `PropertyBox` value as JSON. At the endpoint resource level, the `PropertyBox` type input parameter of the `serialize` method is annotated with `@PropertySetRef` in order to declare the property set to use to deserialize the property box from JSON

③ Perform a `GET` request for a JSON serialized `PropertyBox` value, providing the `PropertySet` to use for deserialization as a `Context` thread-bound resource

4.2.2. JAX-RS integration configuration

The following configuration properties are available to tune or disable the JAX-RS integration features for *Gson* and `PropertyBox` support:

- `holon.gson.disable-resolver`: If this property is present in JAX-RS application properties, the Gson **ContextResolver** auto-configuration is disabled
- `holon.gson.disable-autoconfig`: If this property is present in JAX-RS application properties, all the Gson and `PropertyBox` JSON serialization/deserialization features will be disabled
- `holon.jaxrs.json.pretty-print`: If `true`, enables *pretty printing* of serialized JSON

4.3. Spring integration

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-gson-spring</artifactId>
<version>5.0.4</version>
```

The SpringGsonConfiguration utility class can be used to configure a Spring `RestTemplate`, ensuring that a `GsonHttpMessageConverter` is registered and bound to a `Gson` instance correctly configured for `PropertyBox` JSON serialization/deserialization.

```
class Config {

  @Bean
  public RestTemplate restTemplate() {
    RestTemplate rt = new RestTemplate();
    SpringGsonConfiguration.configure(rt); ①
    return rt;
  }

}
```

① Configure the `RestTemplate`

4.4. Spring Boot integration

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-gson-spring</artifactId>
<version>5.0.4</version>
```

The `GsonAutoConfiguration` Spring Boot *auto-configuration* class is provided to automatically

configure a `Gson` instance **singleton bean**, correctly configured for `PropertyBox` JSON serialization/deserialization.

This way, the `RestTemplate` instances obtained through the `RestTemplateBuilder` Spring Boot builder will be automatically pre-configured with a `Gson` message converter with `PropertyBox` support.

The `Gson` auto-configuration is triggered only if a `Gson` type bean is not already registered in Spring context.

To disable this auto-configuration feature the `GsonAutoConfiguration` class can be excluded:

```
@EnableAutoConfiguration(exclude={GsonAutoConfiguration.class})
```

5. Jackson integration

A `JsonSerializer` and a `JsonDeserializer` are provided to configure PropertyBox serialization to *JSON* format and deserialization from *JSON* format for a Jackson `ObjectMapper`.

5.1. Configuration and use

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-jackson</artifactId>
<version>5.0.4</version>
```

To enable `PropertyBox` handling in Jackson, registering the suitable `JsonSerializer` and a `JsonDeserializer` pair, the JacksonConfiguration utility class can be used:

```
ObjectMapper mapper = new ObjectMapper(); ①

JacksonConfiguration.configure(mapper); ②
```

① Get or create a Jackson `ObjectMapper`

② Configure the `ObjectMapper` for `PropertyBox` support

With a properly configured `ObjectMapper` instance, a `PropertyBox` can be serialized and deserialized just like any another object. As described in Dealing with properties and `PropertySet`, you need to declare the `PropertySet` to use as a *Context* resource at `PropertyBox` deserialization time.

```
final static PathProperty<Long> ID = PathProperty.create("id", Long.class);
final static PathProperty<String> DESCRIPTION = PathProperty.create("description",
String.class);

final static PropertySet<?> PROPERTY_SET = PropertySet.of(ID, DESCRIPTION);

public void serializeAndDeserialize() throws JsonProcessingException {
  ObjectMapper mapper = new ObjectMapper();
  JacksonConfiguration.configure(mapper); ①

  PropertyBox box = PropertyBox.builder(PROPERTY_SET).set(ID, 1L).set(DESCRIPTION,
"Test").build(); ②

  // serialize
  String json = mapper.writer().writeValueAsString(box); ③
  // deserialize
  box = PROPERTY_SET.execute(() -> mapper.reader().forType(PropertyBox.class)
.readValue(json)); ④
}
```

① Use a properly configured `ObjectMapper` instance

② Build a `PropertyBox` using `PROPERTY_SET` as property set

③ Serialize the `PropertyBox` to JSON.

④ Deserialize back the JSON definition to a `PropertyBox` instance using `PROPERTY_SET` as property set, declaring it as thread-bound *Context* resource through the `execute(…)` method

In the example above, the `PropertyBox` instance will be serialized as a JSON object like this:

```
{
  "id": 1,
  "description": "Test"
}
```

5.2. JAX-RS integration

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-jackson-jaxrs</artifactId>
<version>5.0.4</version>
```

A set of JAX-RS extension classes are provided to configure `PropertyBox` JSON support in JAX-RS using *Jackson*.

The JAX-RS *Jackson* extensions relies on standard `ContextResolver` type to provide a properly

configured `ObjectMapper` instance to use to perform JSON serialization and deserialization.

To setup *Jackson* for `PropertyBox` handling in JSON message type, the JacksonFeature JAX-RS `Feature` can be registered in the JAX-RS application.

If you use Jersey or Resteasy as JAX-RS implementation, there is no need to explicitly register the `JacksonFeature`, just ensure the `holon-jackson` jar is in classpath and the *Jackson* support will be configured automatically, leveraging Jersey *AutoDiscoverable* and Resteasy Java Service extensions features.

5.2.1. `PropertyBox` deserialization

When a `PropertyBox` is used as a JAX-RS resource method **parameter** (for methods which declare to consume `application/json` media type), the JSON deserialization of the input into a `PropertyBox` instance needs to know the `PropertySet` to use in order to create the property box. For this purpose, the `@PropertySetRef` annotation can be used at method parameter level to declare the `PropertySet` instance to use to deserialize the property box.

The PropertySetRef annotation allows to declare the `PropertySet` instance as the `public static` **field** of a given class, which must be specified in the `value()` annotation attribute. If more than one `public static` field of `PropertySet` type is present in the declared class, the `field()` annotation attribute can be used to specify the right field name.

```
final static PathProperty<Integer> CODE = PathProperty.create("code", Integer.class);
final static PathProperty<String> NAME = PathProperty.create("name", String.class);

final static PropertySet<?> PROPERTYSET = PropertySet.of(CODE, NAME);

// JAX-RS example endpoint
@Path("test")
public static class Endpoint {

  @PUT
  @Path("serialize")
  @Consumes(MediaType.APPLICATION_JSON)
  public Response create(@PropertySetRef(value = ExampleJackson.class, field =
"PROPERTYSET") PropertyBox data) {
    return Response.accepted().build();
  }

  @GET
  @Path("deserialize")
  @Produces(MediaType.APPLICATION_JSON)
  public PropertyBox getData() {
    return PropertyBox.builder(PROPERTYSET).set(CODE, 1).set(NAME, "Test").build();
  }

}

public void jaxrs() {
  Client client = ClientBuilder.newClient(); ①

  PropertyBox box1 = PropertyBox.builder(PROPERTYSET).set(CODE, 1).set(NAME, "Test")
.build();

  client.target("https://host/test/serialize").request().put(Entity.entity(box1,
MediaType.APPLICATION_JSON)); ②

  PropertyBox box2 = PROPERTYSET
      .execute(() -> client.target("https://host/test/deserialize").request().get
(PropertyBox.class)); ③

}
```

① Create a JAX-RS `Client`

② Perform a `PUT` request providing a `PropertyBox` value as JSON. At the endpoint resource level, the `PropertyBox` type input parameter of the `serialize` method is annotated with `@PropertySetRef` in order to declare the property set to use to deserialize the property box from JSON

③ Perform a `GET` request for a JSON serialized `PropertyBox` value, providing the `PropertySet` to use for deserialization as a `Context` thread-bound resource

5.2.2. JAX-RS integration configuration

The following configuration properties are available to tune or disable the JAX-RS integration features for *Jackson* `PropertyBox` support:

- `holon.jackson.disable-resolver`: If this property is present in JAX-RS application properties, the Jackson **ContextResolver** auto-configuration is disabled
- `holon.jackson.disable-autoconfig`: If this property is present in JAX-RS application properties, all the Jackson `ObjectMapper` and `PropertyBox` JSON serialization/deserialization features will be disabled
- `holon.jaxrs.json.pretty-print`: If `true`, enables *pretty printing* of serialized JSON

5.3. Spring integration

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-jackson-spring</artifactId>
<version>5.0.4</version>
```

The SpringJacksonConfiguration utility class can be used to configure a Spring `RestTemplate`, ensuring that a `MappingJackson2HttpMessageConverter` is registered and bound to a `ObjectMapper` instance correctly configured for `PropertyBox` JSON serialization/deserialization.

```
class Config {

  @Bean
  public RestTemplate restTemplate() {
    RestTemplate rt = new RestTemplate();
    SpringJacksonConfiguration.configure(rt); ①
    return rt;
  }

}
```

① Configure the `RestTemplate`

5.4. Spring Boot integration

Maven coordinates:

```
<groupId>com.holon-platform.json</groupId>
<artifactId>holon-jackson-spring</artifactId>
<version>5.0.4</version>
```

The `JacksonAutoConfiguration` Spring Boot *auto-configuration* class is provided to automatically configure an `ObjectMapper` for `PropertyBox` JSON serialization/deserialization. An `ObjectMapper` instance must be available as *bean* in the Spring context.

This way, the `RestTemplate` instances obtained through the `RestTemplateBuilder` Spring Boot builder will be automatically pre-configured with a JSON message converter with `PropertyBox` support.

To disable this auto-configuration feature the `JacksonAutoConfiguration` class can be excluded:

```
@EnableAutoConfiguration(exclude={JacksonAutoConfiguration.class})
```

6. Loggers

By default, the Holon platform uses the SLF4J API for logging. The use of SLF4J is optional: it is enabled when the presence of SLF4J is detected in the classpath. Otherwise, logging will fall back to JUL (`java.util.logging`).

The logger names for the **JSON** module are:

- `com.holonplatform.json.gson` for the *Gson* integration classes
- `com.holonplatform.json.jackson` for the *Jackson* integration classes

7. System requirements

7.1. Java

The Holon Platform JSON module requires Java **8** or higher.