



Holon

PLATFORM

Holon JAX-RS integration

Version 5.0.2

Table of Contents

1. Introduction	1
2. Obtaining the artifacts	1
2.1. Using the Platform BOM	2
3. PropertyBox serialization and deserialization support	2
3.1. JSON media type	2
3.2. Form/URLencoded media type	3
4. JAX-RS RestClient implementation	4
4.1. Getting started	5
5. JAX-RS Server	6
5.1. Authentication	6
5.1.1. Realm configuration	7
5.1.2. Authentication schemes	7
5.1.3. Example	7
5.2. Authorization	8
6. Spring Boot integration	9
6.1. JAX-RS Client	9
6.2. Jersey	11
6.2.1. Authentication and authorization	11
6.3. Resteasy	12
6.3.1. Configuration	12
6.3.2. ResteasyConfigCustomizer	12
6.3.3. Configuration properties	12
6.3.4. Authentication and authorization	13
6.4. Spring Boot starters	13
6.4.1. JAX-RS client	14
6.4.2. JAX-RS server	14
7. Swagger integration	15
7.1. PropertyBox support	15
7.2. PropertyBox model definition	15
7.3. Spring Boot integration	16
7.3.1. Swagger API documentation endpoints configuration using properties	16
7.3.2. Swagger configuration properties	17
7.3.3. Swagger API documentation endpoints auto-configuration	19
8. Loggers	19
9. System requirements	20
9.1. Java	20

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 **JAX-RS** module provides support, components and configuration helpers concerning the [JAX-RS](#) - Java API for RESTful Web Service standard.

The module provides **JAX-RS** implementations and integrations for platform foundation components and structures, such as the [RestClient](#) API, server-side authentication and authorization using a [Realm](#) and a complete [Swagger OpenAPI](#) support for data containers such as [PropertyBox](#).

Furthermore, the module makes available a set of **auto-configuration** features, both for the *JAX-RS* ecosystem and for the **Spring** and **Spring Boot** world.

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.jaxrs</groupId>
<artifactId>holon-jaxrs-bom</artifactId>
<version>5.0.2</version>
```

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

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.holon-platform.jaxrs</groupId>
      <artifactId>holon-jaxrs-bom</artifactId>
      <version>5.0.2</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. **PropertyBox** serialization and deserialization support

The [PropertyBox](#) data container serialization and deserialization support for JAX-RS compliant servers and clients is available using the following *media types*:

- `application/json` - see [JSON media type](#)
- `application/x-www-form-urlencoded` - see [Form/URLencoded media type](#)

3.1. JSON media type

The **JSON** serialization and deserialization support for **PropertyBox** is provided by the [Holon Platform JSON module](#). Both [Jackson](#) and [Gson](#) JSON processors are supported.

Just ensure the suitable artifact is present in classpath to auto-configure the required *providers* and *context resolvers* for a JAX-RS server and/or client to enable **JSON** serialization and deserialization support for **PropertyBox**:

- `com.holon-platform.json / holon-jackson-jaxrs` for JSON support using **Jackson**
- `com.holon-platform.json / holon-gson-jaxrs` for JSON support using **Gson**

See the [Holon Platform JSON module documentation](#) for details and configuration options.

With the **PropertyBox** *JSON* support enabled, you can write JAX-RS endpoints like this:

```

@Path("propertybox")
public static class JsonEndpoint {

    @GET
    @Path("get")
    @Produces(MediaType.APPLICATION_JSON)
    public PropertyBox getPropertyBox() { ❶
        return PropertyBox.builder(PROPERTY_SET).set(A_PROPERTY, 1).build();
    }

    @GET
    @Path("getList")
    @Produces(MediaType.APPLICATION_JSON)
    public List<PropertyBox> getPropertyBoxList() { ❷
        return Collections.singletonList(PropertyBox.builder(PROPERTY_SET).set(A_PROPERTY,
1).build());
    }

    @PUT
    @Path("put")
    @Consumes(MediaType.APPLICATION_JSON)
    public Response putPropertyBox(@PropertySetRef(ExamplePropertyBox.class) PropertyBox
data) { ❸
        return Response.accepted().build();
    }

}

```

- ❶ A **GET** endpoint method which returns a JSON-encoded **PropertyBox** instance
- ❷ A **GET** endpoint method which returns a JSON-encoded **PropertyBox** instances **List**
- ❸ A **PUT** endpoint method which accepts a JSON-encoded **PropertyBox** as body parameter. The **@PropertySetRef** annotation is used to specify the **PropertySet** to be used to decode the **PropertyBox** from JSON

3.2. Form/URLencoded media type

The **application/x-www-form-urlencoded** media type for **PropertyBox** serialization and deserialization is supported by default and auto-configured when one of the artifacts of the **JAX-RS** is present in classpath.

You can explicitly configure the **application/x-www-form-urlencoded** media type support in a JAX-RS server or client registering the **FormDataPropertyBoxFeature**.



Only **simple data types** (Strings, Numbers, Booleans, Enums and Dates) are supported for **PropertyBox** serialization and deserialization using the **application/x-www-form-urlencoded** media type, so you cannot use complex property values (such as JavaBeans) as **PropertyBox** property values. The **JSON** media type is strongly recommended as **PropertyBox** data interchange format in a JAX-RS environment.

With the **PropertyBox** for the *form/urlencoded* format support enabled, you can write JAX-RS endpoints like this:

```
@Path("propertybox")
public static class FormDataEndpoint {

    @POST
    @Path("post")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public Response postPropertyBox(@PropertySetRef(ExamplePropertyBox.class)
PropertyBox data) { ①
        return Response.ok().build();
    }
}
```

- ① A **POST** endpoint method which accepts a JSON-encoded **PropertyBox** as body parameter. The **@PropertySetRef** annotation is used to specify the **PropertySet** to be used to decode the **PropertyBox** from **application/x-www-form-urlencoded** data

4. JAX-RS **RestClient** implementation

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-client</artifactId>
<version>5.0.2</version>
```

This artifact makes available a **JAX-RS** standard implementation of the platform **RestClient** API, a Java client to deal with *RESTful web services* APIs using the **HTTP** protocol.

The **RestClient** interface provides a fluent *builder* to compose and execute *RESTful web services* invocations, using *template* variable substitution, supporting base authentication methods, common headers configuration and request entities definition.

The **RestClient** API ensures support for the **PropertyBox** data container out-of-the-box.



See [RestClient documentation](#) for information about the **RestClient** configuration and API.

4.1. Getting started

To obtain a **JAX-RS `RestClient`** builder, the `create()` method of the `JaxrsRestClient` interface can be used, either specifying the **JAX-RS `Client`** instance or using the default client builder implementation class provided by the JAX-RS implementation provider.

Furthermore, a `RestClientFactory` is automatically registered to provide a `JaxrsRestClient` implementation using default `RestClient` creation methods.

RestClient example

```
final PathProperty<Integer> ID = PathProperty.create("id", Integer.class);
final PathProperty<String> NAME = PathProperty.create("name", String.class);

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

RestClient client = JaxrsRestClient.create() ①
    .defaultTarget(new URI("https://host/api")); ②

client = RestClient.create(JaxrsRestClient.class.getName()); ③

client = RestClient.create(); ④

client = RestClient.forTarget("https://host/api"); ⑤

Optional<TestData> testData = client.request().path("data/{id}").resolve("id", 1) ⑥
    .accept(MediaType.APPLICATION_JSON).getForEntity(TestData.class);

Optional<PropertyBox> box = client.request().path("getbox") ⑦
    .propertySet(PROPERTY_SET).getForEntity(PropertyBox.class);

HttpResponse<PropertyBox> response = client.request().path("getbox") ⑧
    .propertySet(PROPERTY_SET).get(PropertyBox.class);

List<PropertyBox> boxes = client.request().path("getboxes") ⑨
    .propertySet(PROPERTY_SET).getAsList(PropertyBox.class);

PropertyBox postBox = PropertyBox.builder(PROPERTY_SET).set(ID, 1).set(NAME, "Test")
    .build();

HttpResponse<Void> postResponse = client.request().path("postbox") ⑩
    .post(RequestEntity.json(postBox));
```

- ① Create a **JAX-RS `RestClient`** using default JAX-RS Client
- ② Setup a *default target*, i.e. the base `URI` which will be used for all the invocations made with this `RestClient` instance
- ③ Create a `RestClient` specifying the `JaxrsRestClient` type
- ④ Create a `RestClient` using default factories

- ⑤ Create a `RestClient` using default factories and set a default base URI
- ⑥ Get a generic JSON response using a `template` variable and map it into a `TestData` bean
- ⑦ Get a `PropertyBox` type JSON response entity content using given `PROPERTY_SET`
- ⑧ Get a `PropertyBox` type JSON response using given `PROPERTY_SET`
- ⑨ Get a `List` of `PropertyBox` JSON response entity content using given `PROPERTY_SET`
- ⑩ Post a `PropertyBox` instance as JSON

5. JAX-RS Server

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-server</artifactId>
<version>5.0.2</version>
```

This artifact deals with JAX-RS server resources [Authentication](#) and [Authorization](#), using platform foundation APIs, such as [Realm](#).

5.1. Authentication

The [AuthenticationFeature](#) can be used in a JAX-RS server to enable authentication support using the core [Authenticate](#) annotation.

The feature is automatically registered for **Jersey** and **Resteasy** server runtimes using Java Service extensions and auto-discoverable. To explicitly disable the feature, the `holon.jaxrs.server.disable-authentication` property name can be used, registering it as a server configuration property name (with an arbitrary not null value).

When this feature is enabled, the standard JAX-RS `SecurityContext` of every request will be replaced with an [AuthContext](#) compatible implementation, which will be used to perform authentication when required.



See [Authentication and authorization](#) documentation for information about the platform authentication and authorization architecture, API and components.

The JAX-RS resource classes and/or methods annotated with `@Authenticate` will be protected from unauthorized access, performing client authentication using the `AuthContext` security context setted up by this feature as described above.

When authentication informations provided by a client for an `@Authenticate` annotated resource are missing or invalid, a `401 - Unauthorized` status response is returned, including a `WWW_AUTHENTICATE` header for each allowed authentication scheme, if any.

5.1.1. Realm configuration

The feature relies on a **Realm** to perform authentication operations, which must be available either:

- From a suitable JAX-RS **ContextResolver** which returns the **Realm** instance
- Or as a platform standard **Context** resource with default **Realm** context key (i.e. `com.holonplatform.auth.Realm`)

5.1.2. Authentication schemes

Allowed authentication **schemes** can be specified using the `schemes()` attribute of the `@Authenticate` annotation attribute. If any scheme is specified for a resource, a scheme-matching **AuthenticationTokenResolver** must be registered in **Realm** to perform authentication with given scheme.

See **MessageAuthenticator** for information about *message authenticators* and builtin authenticators for HTTP schemes like **Basic** and **Bearer**.

5.1.3. Example

```
@Authenticate(schemes = HttpHeaders.SCHEME_BASIC) ①
@Path("protected")
class ProtectedResource {

    @GET
    @Path("test")
    @Produces(MediaType.TEXT_PLAIN)
    public String test() {
        return "test";
    }
}

@Path("semiprotected")
class SemiProtectedResource { ②

    @GET
    @Path("public")
    @Produces(MediaType.TEXT_PLAIN)
    public String publicMethod() { ③
        return "public";
    }

    @Authenticate(schemes = HttpHeaders.SCHEME_BASIC) ④
    @GET
    @Path("protected")
    @Produces(MediaType.TEXT_PLAIN)
    public String protectedMethod() {
        return "protected";
    }
}
```

```

}

// configuration
public void configureJaxrsApplication() {

    AccountProvider provider = id -> { ⑤
        // a test provider wich always returns an Account with given id and s3cr3t as
        password
        return Optional.ofNullable(Account.builder(id).credentials(Credentials.builder()
            .secret("s3cr3t").build())
            .enabled(true).build());
    };

    Realm realm = Realm.builder() ⑥
        .resolver(AuthenticationToken.httpBasicResolver()) ⑦
        .authenticator(Account.authenticator(provider)) ⑧
        .withDefaultAuthorizer().build();

    ContextResolver<Realm> realmContextResolver = new ContextResolver<Realm>() { ⑨

        @Override
        public Realm getContext(Class<?> type) {
            return realm;
        }
    };

    register(realmContextResolver); ⑩
}

```

- ① JAX-RS endpoint resource protected using `@Authenticate` and `Basic` HTTP authentication scheme
- ② JAX-RS endpoint resource with only one protected method
- ③ This method is not protected
- ④ Only this method of the resource is protected using `@Authenticate` and `Basic` HTTP authentication scheme
- ⑤ `AccountProvider` to provide available `Account` s to the Realm
- ⑥ Build a `Realm` to be used for resource access authentication
- ⑦ Add a *resolver* for HTTP `Basic` scheme authentication messages
- ⑧ Set the realm *authenticator* using the previously defined `AccountProvider`
- ⑨ Create a JAX-RS `ContextResolver` to provide the `Realm` instance to use
- ⑩ Register the Realm `ContextResolver` in JAX-RS application (for example, using a Jersey `ResourceConfig`)

5.2. Authorization

When a `SecurityContext` is setted up, for example using the `Authentication` feature, it can be used to

check if an account is authenticated and perform role-based access control.

For example, to use standard `javax.annotation.security` annotations on resource classes for role-based access control, you can:

- In **Jersey**, register the standard `RolesAllowedDynamicFeature` in server resources configuration.
- In **Resteasy**, activate the role-based security access control setting a servlet the context parameter `resteasy.role.based.security` to `true`.

6. Spring Boot integration

6.1. JAX-RS Client

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-client</artifactId>
<version>5.0.2</version>
```

This artifact provides a Spring Boot auto-configuration class to automatically register a `JaxrsClientBuilder` bean, which can be used to obtain a configured JAX-RS `Client` instance.

To customize the JAX-RS `ClientBuilder` used to obtain the client instances, the `JaxrsClientCustomizer` interface can be used: any Spring bean which implement the `JaxrsClientCustomizer` interface will be auto-detected and the `customize(ClientBuilder clientBuilder)` method will be invoked when a `ClientBuilder` is created.

To replace the default `ClientBuilder` instance lookup/creation strategy, a `JaxrsClientBuilderFactory` bean type can be declared in Spring context, which will be used by the `JaxrsClientBuilder` to create a new `ClientBuilder` instance.

Furthermore, a `RestClient` factory is automatically registered to use the `JaxrsClientBuilder` (if available) to obtain the JAX-RS `Client` instance to be used by a JAX-RS `RestClient` created through the `RestClient.create()` method.

For example, given a Spring Boot application with the following configuration:

```

@SpringBootApplication
static class Application {

    @Bean
    public JaxrsClientCustomizer propertyCustomizer() { ①
        return cb -> cb.property("test.jaxrs.client.customizers", "test");
    }

    @Bean
    public JaxrsClientCustomizer sslCustomizer() throws KeyManagementException,
        NoSuchAlgorithmException { ②
        // setup a SSLContext with a "trust all" manager
        final SSLContext sslcontext = SSLContext.getInstance("TLS");
        sslcontext.init(null, new TrustManager[] { new X509TrustManager() {
            @Override
            public void checkClientTrusted(X509Certificate[] arg0, String arg1) throws
                CertificateException {
            }

            @Override
            public void checkServerTrusted(X509Certificate[] arg0, String arg1) throws
                CertificateException {
            }

            @Override
            public X509Certificate[] getAcceptedIssuers() {
                return new X509Certificate[0];
            }
        } }, new java.security.SecureRandom());

        return cb -> {
            // customize ClientBuilder
            cb.sslContext(sslcontext).hostnameVerifier((s1, s2) -> true);
        };
    }

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

```

① Add a **JaxrsClientCustomizer** which registers a property in JAX-RS **ClientBuilder**

② Add a **JaxrsClientCustomizer** which setup the **ClientBuilder** to use a **SSLContext** with a *trust all* manager and dummy host name verifier

A JAX-RS Client, configured according to the declared customizers, can be obtained as follows:

```
@Autowired
private JaxrsClientBuilder clientBuilder;

private void getClient() {
    Client jaxrsClient = clientBuilder.build(); ①

    RestClient restClient = RestClient.create(); ②
}
```

① Use the `JaxrsClientBuilder` to obtain a new JAX-RS `Client` instance

② Use the `RestClient.create()` static method to obtain a `RestClient` which uses a JAX-RS `Client` obtained from the `JaxrsClientBuilder`

6.2. Jersey

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-jersey</artifactId>
<version>5.0.2</version>
```

This artifact provides a `Jersey` Spring Boot auto-configuration class to automatically register any Spring context **bean** annotated with `@Path` or `@Provider` JAX-RS annotations as a server resource, using the default `ResourceConfig` Jersey configuration bean.

If custom `ResourceConfig` type bean is not available in Spring context, a **default one** will be automatically created.



For `@Provider` annotated bean classes, only **singleton** scoped beans are allowed.

The `holon.jersey.bean-scan` Spring boot application property can be used, setting it to `false`, to disable automatic bean resources scan and registration.

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

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

6.2.1. Authentication and authorization

When a `Realm` type bean is detected in Spring context, the JAX-RS server is automatically configured to support **authentication**, relying on `@Authenticate` annotation (see [Authentication](#)), and **authorization**, relying on `javax.annotation.security.*` annotations.

The auto configuration class performs the following operations:

- Registers a `ContextResolver` providing the `Realm` bean instance

- Registers the `AuthenticationFeature`
- Registers the Jersey `RolesAllowedDynamicFeature`

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

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

6.3. Resteasy

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-resteasy</artifactId>
<version>5.0.2</version>
```

This artifact provides a `Resteasy` Spring Boot auto-configuration classes to **automatically setup a Resteasy JAX-RS server implementation** with Spring integration enabled, providing configuration through Spring Boot application properties.

6.3.1. Configuration

The `ResteasyConfig` class, which extends default JAX-RS `Application` class, can be used to register the JAX-RS resources, similarly to the `ResourceConfig` Jersey configuration class.

The `ResteasyConfig` must be declared as a singleton Spring bean to be used by the Resteasy auto-configuration classes. If a `ResteasyConfig` type bean is not available, a **default one** will be automatically created.

Just like the `Jersey` auto-configuration artifact, this module automatically register any Spring context **bean** annotated with `@Path` or `@Provider` JAX-RS annotations as a server resource.



For `@Provider` annotated bean classes, only **singleton** scoped beans are allowed.

The Resteasy JAX-RS application path can be defined either using the `ApplicationPath` annotation on the `ResteasyConfig` bean class or through the `holon.resteasy.application-path` configuration property. See [Configuration properties](#) for a list of available configuration properties.

6.3.2. ResteasyConfigCustomizer

Any Spring bean which implements the `ResteasyConfigCustomizer` interface, is automatically discovered and its `customize` method is called, allowing to customize the `ResteasyConfig` instance before it is used.

6.3.3. Configuration properties

The `ResteasyConfigurationProperties` lists the configuration properties (with the `holon.resteasy`

prefix) which can be used to setup the Resteasy auto-configuration.



Just like any other Spring Boot configuration property, the `holon.resteasy.*` properties can be specified in your `application.properties` / `application.yml` file or as command line switches.

Name	Default value	Meaning
<code>holon.resteasy. application-path</code>	<i>no default</i>	Path that serves as the base URI for the application. Overrides the value of <code>@ApplicationPath</code> if specified
<code>holon.resteasy. type</code>	<code>servlet</code>	Resteasy integration type: <code>servlet</code> or <code>filter</code>
<code>holon.resteasy. filter.order</code>	<code>0</code>	Resteasy filter chain order when integration type is <code>filter</code>
<code>holon.resteasy. servlet.load-on-startup</code>	<code>-1</code>	Load on startup priority of the Resteasy servlet when integration type is <code>servlet</code>
<code>holon.resteasy. init.</code>	<i>no default</i>	Init parameters to pass to Resteasy via the servlet or filter

6.3.4. Authentication and authorization

When a `Realm` type bean is detected in Spring context, the JAX-RS server is automatically configured to support **authentication**, relying on `@Authenticate` annotation (see [Authentication](#)), and **authorization**, relying on `javax.annotation.security.*` annotations.

The auto configuration class performs the following operations:

- Registers a `ContextResolver` providing the `Realm` bean instance
- Registers the `AuthenticationFeature`
- Set the `resteasy.role.based.security` context init parameter to `true`

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

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

6.4. Spring Boot starters

The following *starter* artifacts are available to provide a quick JAX-RS server and/or client application setup using the Maven dependency system.

All the available *starters* include the default Holon *core* Spring Boot starters (see the documentation for further information) and the base Spring Boot starter (`spring-boot-starter`).

The **Jersey** *starters* include the default Spring Boot Jersey starter (`spring-boot-starter-jersey`).

The **Resteasy starters** include the default Spring Boot Web starter ([spring-boot-starter-web](#)).

The Maven **group id** for all the JAX-RS *starters* is `com.holon-platform.jaxrs`. So you can declare a *starter* in you `pom` dependencies section like this:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-starter-xxx</artifactId>
<version>5.0.2</version>
```

6.4.1. JAX-RS client

Artifact id	Description
<code>holon-starter-jersey-client</code>	JAX-RS <i>client</i> starter using Jersey and Jackson as JSON provider
<code>holon-starter-jersey-client-gson</code>	JAX-RS <i>client</i> starter using Jersey and Gson as JSON provider
<code>holon-starter-resteasy-client</code>	JAX-RS <i>client</i> starter using Resteasy and Jackson as JSON provider
<code>holon-starter-resteasy-client-gson</code>	JAX-RS <i>client</i> starter using Resteasy and Gson as JSON provider

6.4.2. JAX-RS server

Artifact id	Description
<code>holon-starter-jersey</code>	JAX-RS <i>server</i> starter using Jersey , Tomcat as embedded servlet container and Jackson as JSON provider
<code>holon-starter-jersey-gson</code>	JAX-RS <i>server</i> starter using Jersey , Tomcat as embedded servlet container and Gson as JSON provider
<code>holon-starter-jersey-undertow</code>	JAX-RS <i>server</i> starter using Jersey , Undertow as embedded servlet container and Jackson as JSON provider
<code>holon-starter-jersey-undertow-gson</code>	JAX-RS <i>server</i> starter using Jersey , Undertow as embedded servlet container and Gson as JSON provider
<code>holon-starter-resteasy</code>	JAX-RS <i>server</i> starter using Resteasy , Tomcat as embedded servlet container and Jackson as JSON provider
<code>holon-starter-resteasy-gson</code>	JAX-RS <i>server</i> starter using Resteasy , Tomcat as embedded servlet container and Gson as JSON provider
<code>holon-starter-resteasy-undertow</code>	JAX-RS <i>server</i> starter using Resteasy , Undertow as embedded servlet container and Jackson as JSON provider

Artifact id	Description
holon-starter-resteasy-undertow-gson	JAX-RS <i>server</i> starter using Resteasy , Undertow as embedded servlet container and Gson as JSON provider

7. Swagger integration

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-swagger</artifactId>
<version>5.0.2</version>
```

This artifact provides a complete integration with [Swagger OpenAPI Specification](#), to configure API listing JAX-RS endpoints and to handle the [PropertyBox](#) data container type as a Swagger *Model* definition.

7.1. PropertyBox support

When the [holon-jaxrs-swagger](#) artifact is in classpath, the Swagger JAX-RS engine is automatically configured to support the [PropertyBox](#) type as API parameter or return type, but to ensure a proper [PropertyBox](#) handling the internal *SwaggerContext* must be setted up. This can be done either by:

- Using the [SwaggerConfiguration](#) Swagger [BeanConfig](#) extension class, instead of the standard [BeanConfig](#) class
- Or providing the [SwaggerContextListener](#) Swagger [ReaderListener](#) class as a Swagger scanned resource class

7.2. PropertyBox model definition

A [PropertyBox](#) type API parameter or return type is translated in a regular Swagger *object* definition, listing all the **properties** of the [PropertyBox](#) property-set which implement the [Path](#) interface, using the [Path](#) name as *object* attribute name and the property type (adapted to a standard JSON type) as attribute type.

The [x-holon-model-type](#) extension property is added to each Swagger [PropertyBox](#) type object definition, with the [com.holonplatform.core.property.PropertyBox](#) value.

The [PropertySetRef](#) annotation has to be used in JAX-RS resource methods for [PropertyBox](#) type parameters or return types to declare the **property set** to be used to serialize a [PropertyBox](#) declaration.

To create a regular Swagger **Model definition**, listed in the *definitions* section of the Swagger specification and referenced by name by API definitions, the [ApiPropertySetModel](#) annotation can be used in conjunction with the [@PropertySetRef](#) annotation, using the annotation [value\(\)](#) attribute to declare the model definition **name** to generate for a [PropertyBox](#) type object.



Ensure to assign the same model name to `PropertyBox` type parameters and return types which are meant to be used with the same property set and to declare different model names for different property sets.

7.3. Spring Boot integration

The `holon-jaxrs-swagger` provides Spring Boot auto-configuration classes to automatically configure **Swagger API listing JAX-RS endpoints** in a Spring Boot application.

The Swagger auto-configuration is triggered when either a `ResourceConfig` type *Jersey* configuration bean or a `ResteasyConfig` type *Resteasy* configuration bean is available in Spring context.

7.3.1. Swagger API documentation endpoints configuration using properties

The Swagger auto-configuration relies on the `holon.swagger.*` configuration properties listed in the `SwaggerConfigurationProperties` class to configure the Swagger API endpoints.



Just like any other Spring Boot configuration property, the `holon.swagger.*` properties can be specified in your `application.properties` / `application.yml` file or as command line switches.

At least the `holon.swagger.resource-package` property, which specifies the Java *package* name to scan to detect the JAX-RS API endpoints, must be set to auto-configure a Swagger API listing endpoint. See [Swagger configuration properties](#) for a list of all available configuration properties.

By default, the `/api-docs` default path is used to expose the API listing endpoint, but can be changed using the `holon.swagger.path` property.

The API listing endpoint supports the `type` query parameter name to obtain the Swagger API definitions either in **JSON**, using the `json` parameter value (the default format), or in **YAML**, using the `yaml` parameter value.

For example, for a configuration like the following:

`application.yml`

```
holon:
  swagger:
    version: "v1"
    title: "Test Swagger API"
    resource-package: "my.api.endpoints"
    path: "docs"
    pretty-print: true
```

The `my.api.endpoints` package is scanned to detect JAX-RS API resources and the API listing endpoint will be available at the `http(s):host/docs` URL.

Configure multiple API listing endpoints

Multiple API listing endpoints configuration is supported using the `holon.swagger.api-groups` configuration property, which accept a list of API *groups*, each bound to a specific API *package* to scan and which can be independently configured using the [Swagger configuration properties](#).

For each API group, an API listing endpoint will be available at the base API listing path (`/api-docs` by default, or the one configured with the `holon.swagger.path` property at the root level) followed by the `group-id` property name. For example: `http(s):host/api-docs/one`.

If a `group-id` is not specified, the `default` group id will be used and the API listing endpoint will be available at the base API listing path.

For example, for a configuration like the following:

application.yml

```
holon:
  swagger:
    version: "v1"
    title: "Test Swagger API"

  api-groups:
    - group-id: "one"
      resource-package: "my.api.endpoints.one"
      description: "The API group 1"
      path: docs/one
    - group-id: "two"
      resource-package: "my.api.endpoints.two"
      description: "The API group 2"
      path: docs/two
```

Two API groups are defined:

- The group 1, bound to the `my.api.endpoints.one` package and for which the API listing endpoint will be available at the `http(s):host/docs/one` URL.
- The group 2, bound to the `my.api.endpoints.two` package and for which the API listing endpoint will be available at the `http(s):host/docs/two` URL.

7.3.2. Swagger configuration properties

Table 1. Common configuration properties

Name	Meaning
<code>holon.swagger.enabled</code>	Whether the Swagger API listing endpoints auto-configuration is enabled. Default is <code>true</code> .
<code>holon.swagger.resourcePackage</code>	The package name to scan to detect API endpoints

Name	Meaning
<i>holon.swagger.</i> path	API listing endpoint path. When at group level, is appended to the base API listing path.
<i>holon.swagger.</i> schemes	API supported protocol schemes list (http , https)
<i>holon.swagger.</i> title	API title
<i>holon.swagger.</i> version	API version
<i>holon.swagger.</i> description	API description
<i>holon.swagger.</i> termsOfServiceUrl	Terms of Service URL
<i>holon.swagger.</i> contact	Contact information
<i>holon.swagger.</i> license	License information
<i>holon.swagger.</i> licenseUrl	License URL
<i>holon.swagger.</i> host	API host name
<i>holon.swagger.</i> pretty-print	Whether to <i>pretty</i> format API listing output (true or false)
<i>holon.swagger.</i> auth-schemes	Enable authentication for the API listing endpoints using the @Authenticate annotation behaviour, specifying the allowed authentication schemes. If only one scheme with the * value is provided, any supported authentication scheme is allowed for authentication.
<i>holon.swagger.</i> security-roles	A list of security roles for API listing access control using the JAX-RS SecurityContext and the @RolesAllowed annotation
<i>holon.swagger.</i> api-groups	Optional API groups. Each group can be configured using the properties listed below.

Table 2. API group configuration properties

Name	Meaning
<i>holon.swagger.api-groups.</i> group-id	API group id, also used as API listing endpoint sub-path is group path is not specified
<i>holon.swagger.api-groups.</i> resourcePackage	The package name to scan to detect API group endpoints
<i>holon.swagger.api-groups.</i> path	API group listing endpoint path
<i>holon.swagger.api-groups.</i> schemes	API group supported protocol schemes list (http , https)
<i>holon.swagger.api-groups.</i> title	API group title
<i>holon.swagger.api-groups.</i> version	API group version
<i>holon.swagger.api-groups.</i> description	API group description
<i>holon.swagger.api-groups.</i> termsOfServiceUrl	Terms of Service URL
<i>holon.swagger.api-groups.</i> contact	Contact information

Name	Meaning
<i>holon.swagger.api-groups</i> . license	License information
<i>holon.swagger.api-groups</i> . licenseUrl	License URL
<i>holon.swagger.api-groups</i> . auth-schemes	Enable authentication for the API group listing using the <code>@Authenticate</code> annotation behaviour, specifying the allowed authentication schemes. If only one scheme with the <code>*</code> value is provided, any supported authentication scheme is allowed for authentication.
<i>holon.swagger.api-groups</i> . security-roles	A list of security roles for API group listing access control using the JAX-RS <code>SecurityContext</code> and the <code>@RolesAllowed</code> annotation

7.3.3. Swagger API documentation endpoints auto-configuration

If the `holon.swagger.resourcePackage` configuration property is not provided and no API groups are defined using the `holon.swagger.apiGroups.*` configuration properties, the Holon Swagger integration module will try to auto-configure the Swagger API listing endpoints, relying on the Swagger `@Api` annotation.

Any JAX-RS `@Path` resource declared as a Spring bean which is annotated with the Swagger `@Api` annotation will be auto-detected and used to obtain the package name to use as Swagger API listing endpoint source.

The default `/api-docs` path is used as the Swagger API listing mapping.

To configure the API listing path and the API information properties, the `ApiDefinition` annotation can be used. The `@ApiDefinition` annotation can be used at package level (annotating a standard `package-info.java` class) or at resource class level, when a single JAX-RS `@Api` resource class is present.

When more than one package which contains valid JAX-RS `@Api` classes is present, the Swagger API listings path must be different for each package, so the `@ApiDefinition` annotation is required to specify the API documentation path for each package.

8. 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 **JAX-RS** module are:

- `com.holonplatform.jaxrs` base JAX-RS module logger
- `com.holonplatform.jaxrs.swagger` for the *Swagger* integration classes

9. System requirements

9.1. Java

The Holon Platform JSON module requires [Java 8](#) or higher.

The *JAX-RS* specification version **2.0 or above** is required.

This module is tested against [Jersey](#) version **2.x** and [Resteasy](#) version **3.x**.