Holon Platform JAX-RS Module - Reference manual

Table of Contents

1. Introduction . . . 2
   1.1. Sources and contributions . . . 3
2. Obtaining the artifacts . . . 3
   2.1. Using the Platform BOM . . . 4
3. What's new in version 5.3.x . . . 4
4. What's new in version 5.2.x . . . 4
5. What's new in version 5.1.x . . . 5
6. `PropertyBox` serialization and deserialization support . . . 5
   6.1. JSON media type . . . 5
   6.2. Form/URLencoded media type . . . 6
7. JAX-RS `RestClient` implementation . . . 7
   7.1. Getting started . . . 8
8. JAX-RS `AsyncRestClient` implementation . . . 9
   8.1. Getting started . . . 9
9. JAX-RS `ReactiveRestClient` implementation . . . 11
   9.1. Getting started . . . 11
10. JAX-RS Server . . . 13
   10.1. Disable the Authentication feature . . . 13
   10.2. Authentication . . . 13
      10.2.1. JAX-RS `Realm` configuration . . . 14
      10.2.2. Authentication schemes . . . 15
   10.3. Using `AuthenticationInspector` with JAX-RS `SecurityContext` . . . 15
   10.4. Authorization . . . 16
      10.4.1. Example . . . 17
   10.5. JAX-RS HttpRequest . . . 18
11. Spring Security integration . . . 19
   11.1. Feature configuration . . . 20
12. Spring Boot integration . . . 20
   12.1. JAX-RS Client . . . 20
   12.2. Jersey . . . 23
      12.2.1. Automatic JAX-RS server resources registration . . . 23
      12.2.2. Handling the `jersey.config.servlet.filter.forwardOn404` configuration property . . . 23
      12.2.3. Authentication and authorization . . . 24
   12.3. Resteasy . . . 24

12.3.1. Configuration . . . 24
12.3.2. Resteasy configuration customization . . . 25
12.3.3. Automatic JAX-RS server resources registration . . . 25
12.3.4. Resteasy configuration properties . . . 25
12.3.5. Authentication and authorization . . . 26
12.4. Spring Boot starters . . . 26
12.4.1. JAX-RS client . . . 26
12.4.2. JAX-RS server . . . 27
13. Swagger / OpenAPI integration . . . 27
13.1. `PropertyBox` data type support . . . 28
13.1.1. `PropertyBox` property set declaration . . . 29
13.1.2. Creating a Swagger Model/Schema definition for a `PropertyBox` type . . . 33
13.2. Using the `ApiReader` to generate the API definition model . . . 37
13.2.1. Swagger version 2 `ApiReader` . . . 37
13.2.2. Swagger/OpenAPI version 3 `ApiReader` . . . 37
13.3. Swagger Spring Boot integration and auto configuration . . . 38
13.3.1. Prerequisites . . . 38
13.3.2. Default API endpoints configuration strategy . . . 39
13.3.3. API definition endpoint path . . . 40
13.3.4. API definition endpoint types . . . 40
13.3.5. API resources scan strategy . . . 41
13.3.6. API and endpoints configuration using application properties . . . 41
13.3.7. API and endpoints configuration using beans . . . 45
13.3.8. Multiple API endpoints configuration . . . 47
13.4. Enabling Swagger V2 and V3 API endpoints simultaneously . . . 53
13.5. Disabling the Swagger API endpoints auto-configuration . . . 54
13.6. `ApiDefinition` annotation deprecation . . . 55
14. Loggers . . . 55
15. System requirements . . . 55
15.1. Java . . . 56

Copyright © 2016-2019

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 Services*.

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.

Regarding the **JSON** data-interchange format, this module uses the Holon JSON module to make available the Holon platform JSON extensions and configuration facilities for JAX-RS endpoints and clients, allowing to seamlessy use Jackson or Gson as JSON providers and provide support for *temporal* types (including the `java.time.*` API) and the `PropertyBox` type out-of-the-box.

The module provides a full support for Swagger and the **OpenAPI specification** including support for the `PropertyBox` type (to be exposed as a Swagger *Model* definition) and for Swagger API listing endpoints (both in *JSON* and *YAML* formats) auto-configuration.

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.

A complete support for the most used JAX-RS implementations (Jersey and Resteasy) is provided, including Resteasy auto-configuration classes for Spring Boot integration.

1.1. Sources and contributions

The Holon Platform **JAX-RS** module source code is available from the GitHub repository https://github.com/holon-platform/holon-jaxrs.

See the repository `README` file for information about:

- The source code structure.
- How to build the module artifacts from sources.
- Where to find the code examples.
- How to contribute to the module development.

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>5.5.0</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>5.5.0</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. What's new in version 5.3.x

- Support for Jersey **2.29** and for the new `jersey-spring5` module, as as a replacement for the `jersey-spring4` module.

4. What's new in version 5.2.x

- A new JAX-RS `AsyncRestClient` implementation to provide asynchronous REST API operations handling, using the standard JVM `CompletionStage` API to handle the operation responses. See JAX-RS `AsyncRestClient` implementation.
- A new `holon-jaxrs-client-reactor` artifact which provides a **reactive** version of the JAX-RS `RestClient`, using the `Mono` and `Flux` Project Reactor types to handle the REST operations response. See JAX-RS `ReactiveRestClient` implementation.
- Full **Swagger/OpenAPI version 3 support** and Spring Boot auto configuration available from the new `holon-jaxrs-swagger-v3` artifact. The Swagger API version 2 Spring Boot integration was completely revised to obtain a seamlessy API configuration and API endpoint configuration for both API specification versions. Furthermore, a new `ApiReader` API is now available. See Swagger / OpenAPI integration.
- Support for JDK 9+ module system using `Automatic-Module-Name`.

- Support for JAX-RS API version **2.1**.

The `holon-jaxrs-swagger` artifact is **deprecated** and should be replaced with the `holon-jaxrs-swagger-v2` artifact. The `holon-jaxrs-swagger` is still available in Holon Platform version 5.2.x, acting as an alias only for backward compatibility purposes.

5. What's new in version 5.1.x

- Improved support for the `java.time. Date and Time API data types when a PropertyBox type is serialized and deserialize as *JSON` in JAX-RS endpoints. See JSON media type.
- The new `JaxrsAuthenticationInspector` API is available in JAX-RS endpoints to inspect current `Authentication` and perform authorization controls using a JAX-RS `SecurityContext`. See Using `AuthenticationInspector` with JAX-RS `SecurityContext`.
- Added support for **Spring Security** based authentication, providing features to integrate the `@Authenticate` annotation based authentication behaviour and using the Spring Security context as authentication handler. See Spring Security integration.
- Improved **Spring Boot** auto-configuration support for *Jersey* and *Resteasy* JAX-RS implementations. See Spring Boot integration.
- Improved Swagger integration and auto-configuration, using Spring Boot application properties for easier configuration. See Swagger Spring Boot integration.

6. `PropertyBox` serialization and deserialization support

The PropertyBox type 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

6.1. JSON media type

The **JSON** serialization and deserialization support for the `PropertyBox` type is provided by the Holon Platform JSON module. Both Jackson and Gson JSON providers are supported.

To learn about `PropertyBox` type mapping strategies and configuration options see the PropertyBox section of the Holon Platform JSON module documentation.

To enable the `PropertyBox` type support for JSON media type, just ensure that a suitable artifact is present in classpath:

- `holon-jackson-jaxrs` to use the **Jackson** library. See Jackson JAX-RS integration for details.

- `holon-gson-jaxrs` to use the **Gson** library. See Gson JAX-RS integration for details.

The auto-configuration facilities provided by this two artifacts allow to automatically register and setup all the required JAX-RS features, both for Jersey and for Resteasy JAX-RS implementations.

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() { ①
    return PropertyBox.builder(PROPERTY_SET).set(A_PROPERTY, 1).build();
  }

  @GET
  @Path("getList")
  @Produces(MediaType.APPLICATION_JSON)
  public List<PropertyBox> getPropertyBoxList() { ②
    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) { ③
    return Response.accepted().build();
  }

}
```

① A `GET` endpoint method which returns a JSON-encoded `PropertyBox` instance

② A `GET` endpoint method which returns a JSON-encoded `PropertyBox` instances `List`

③ 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

6.2. Form/URLencoded media type

The `application/x-www-form-urlencoded` media type for `PropertyBox` serialization and deserialization is supported by default and auto-configured for *Jersey* and *Resteasy* when the `holon-jaxrs-commons` artifact 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 Java beans) as `PropertyBox` property values. The **JSON** media type is strongly recommended as `PropertyBox` data interchange format in a JAX-RS environment.

With the *form/urlencoded* `PropertyBox` type 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

7. JAX-RS `RestClient` implementation

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-client</artifactId>
<version>5.5.0</version>
```

This artifact makes available a **JAX-RS** implementation of the Holon Platform RestClient API, a Java client API to deal with *RESTful web services* 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 type out-of-the-box.

See the RestClient API documentation for information about the `RestClient` configuration and available operations.

7.1. Getting started

To obtain a JAX-RS `RestClient` builder, the `create()` method of the `JaxrsRestClient` interface can be used, either specifying the concrete JAX-RS `javax.ws.rs.client.Client` instance to use or relying on the default JAX-RS `Client` provided by the `javax.ws.rs.client.ClientBuilder` class.

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

See the RestClient factory section of the core documentation for more information about `RestClient` factories.

```
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` API using the 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` API specifying the `JaxrsRestClient` type, to ensure a JAX-RS implementation

of the `RestClient` API is provided

④ Create a `RestClient` API using the default `RestClientFactory` lookup strategy

⑤ Create a `RestClient` API using the default `RestClientFactory` lookup strategy and set a default base URI

⑥ Get a generic JSON response using a *template* variable and map it into a `TestData` type 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` type instance using JSON

8. JAX-RS `AsyncRestClient` implementation

The `holon-jaxrs-client` artifact makes also available an **asynchronous** JAX-RS implementation of the Holon Platform RestClient API, a Java client API to deal with *RESTful web services* using the `HTTP` protocol.

The asynchronous REST client API is represented by the `AsyncRestClient` interface, which 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 standard JVM `CompletionStage` API is used to asynchronously handle the operation responses.

The `AsyncRestClient` API ensures support for the PropertyBox data type out-of-the-box.

8.1. Getting started

To obtain a JAX-RS `AsyncRestClient` builder, the `create()` method of the JaxrsAsyncRestClient interface can be used, either specifying the concrete JAX-RS `javax.ws.rs.client.Client` instance to use or relying on the default JAX-RS `Client` provided by the `javax.ws.rs.client.ClientBuilder` class.

Furthermore, a `AsyncRestClientFactory` is automatically registered to provide a `JaxrsAsyncRestClient` implementation using the default `AsyncRestClient` creation methods.

See the AsyncRestClient factory section of the core documentation for more information about `AsyncRestClient` factories.

```
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);

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

client = AsyncRestClient.create(JaxrsAsyncRestClient.class.getName()); ③

client = AsyncRestClient.create(); ④

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

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

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

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

CompletionStage<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();

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

① Create a JAX-RS `AsyncRestClient` API using the 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 `AsyncRestClient` instance

③ Create a `AsyncRestClient` API specifying the `JaxrsAsyncRestClient` type, to ensure a JAX-RS implementation of the `AsyncRestClient` API is provided

④ Create a `AsyncRestClient` API using the default `AsyncRestClientFactory` lookup strategy

⑤ Create a `AsyncRestClient` API using the default `AsyncRestClientFactory` lookup strategy and set a default base URI

⑥ Get a generic JSON response using a *template* variable and map it into a `TestData` type bean. A `CompletionStage` type result is returned and can be used to asynchronously handle the operation response.

⑦ Get a `PropertyBox` type JSON response entity content using given `PROPERTY_SET`. A `CompletionStage`

type result is returned and can be used to asynchronously handle the operation response.

⑧ Get a `PropertyBox` type JSON response using given `PROPERTY_SET`. A `CompletionStage` type result is returned and can be used to asynchronously handle the operation response.

⑨ Get a `List` of `PropertyBox` JSON response entity content using given `PROPERTY_SET`. A `CompletionStage` type result is returned and can be used to asynchronously handle the operation response.

⑩ Post a `PropertyBox` type instance using JSON

9. JAX-RS `ReactiveRestClient` implementation

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-client-reactor</artifactId>
<version>5.5.0</version>
```

This artifact makes available a **reactive** JAX-RS implementation of the Holon Platform REST client API, using the ReactiveRestClient API.

The `ReactiveRestClient` uses the Project Reactor `Mono` and `Flux` types as REST operation results representation, dependently on the result cardinality.

The `ReactiveRestClient` 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 `ReactiveRestClient` API ensures support for the PropertyBox data type out-of-the-box.

See the ReactiveRestClient API documentation for information about the `ReactiveRestClient` configuration and available operations.

9.1. Getting started

To obtain a JAX-RS `ReactiveRestClient` builder, the `create()` method of the JaxrsReactiveRestClient interface can be used, either specifying the concrete JAX-RS `javax.ws.rs.client.Client` instance to use or relying on the default JAX-RS `Client` provided by the `javax.ws.rs.client.ClientBuilder` class.

Furthermore, a `ReactiveRestClientFactory` is automatically registered to provide a `JaxrsReactiveRestClient` implementation using the default `ReactiveRestClient` creation methods.

See the ReactiveRestClient factory section of the reactor module documentation for more information about the `ReactiveRestClient` factories.

```
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);

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

client = ReactiveRestClient.create(JaxrsReactiveRestClient.class.getName()); ③

client = ReactiveRestClient.create(); ④

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

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

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

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

Flux<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();

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

① Create a JAX-RS `ReactiveRestClient` API using the 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 `ReactiveRestClient` instance

③ Create a `ReactiveRestClient` API specifying the `JaxrsReactiveRestClient` type, to ensure a JAX-RS implementation of the `ReactiveRestClient` API is provided

④ Create a `ReactiveRestClient` API using the default `ReactiveRestClientFactory` lookup strategy

⑤ Create a `ReactiveRestClient` API using the default `ReactiveRestClientFactory` lookup strategy and set a default base URI

⑥ Get a `Mono` type JSON response using a *template* variable and map it into a `TestData` type bean

⑦ Get the `Mono` type `PropertyBox` JSON response entity content using given `PROPERTY_SET`

⑧ Get the `Mono` type `PropertyBox` JSON response using given `PROPERTY_SET`

⑨ Get a `Flux` of `PropertyBox` type JSON response entities using given `PROPERTY_SET`

⑩ Post a `PropertyBox` type instance using JSON

10. JAX-RS Server

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-server</artifactId>
<version>5.5.0</version>
```

The JAX-RS server-side integration deals with the JAX-RS server resources Authentication and Authorization, using platform foundation APIs, such as the Realm API.

The JAX-RS AuthenticationFeature class can be used in a JAX-RS server application to enable the Holon platform **authentication and authorization support** for JAX-RS endpoints, using the platform foundation authentication architecture and APIs.

When the `holon-jaxrs-server` artifact is present in classpath, this feature is automatically registered for **Jersey** and **Resteasy** JAX-RS server runtimes, leveraging on Jersey *AutoDiscoverable* and Resteasy Java Service extensions features.

See the Authentication and authorization documentation for information about the platform authentication and authorization architecture and APIs.

10.1. Disable the Authentication feature

To explicitly disable the JAX-RS `AuthenticationFeature`, the `holon.jaxrs.server.disable-authentication` property name can be used, registering it as a JAX-RS server configuration property name (with an arbitrary not null value).

10.2. Authentication

The JAX-RS authentication integration, through the AuthenticationFeature JAX-RS feature, relies on the Authenticate annotation, which is detected at both JAX-RS resource **class** and **method** level.

The `@Authenticate` annotation can be used to protect JAX-RS resource classes and/or methods from unauthorized access and relies on the Holon platform Realm API to perform actual authentication and authorization operations. For this reason, a `Realm` instance must be configured and available in JAX-RS server context to enable the authentication feature.

See JAX-RS `Realm` configuration for details.

During a JAX-RS request, when the `@Authenticate` annotation is detected on a JAX-RS endpoint resource class and/or method, the authentication and authorization control flow is triggered, which is based on the following strategy:

1. The standard JAX-RS `SecurityContext` of the request is replaced with an AuthContext API

compatible implementation, backed by the configured `Realm`. This `AuthContext` will be used to provide the authenticated *principal*, if available, and to perform role-based authorization controls through the JAX-RS `SecurityContext` API.

2. The request JAX-RS is authenticated using the incoming request message and the request `AuthContext` (and so the `Realm` to which the `AuthContext` is bound), possibly using the authentication *scheme* specified through the `@Authenticate` annotation, if available (see Authentication schemes for details).

   a. If authentication does not succeed (for example when the authentication informations provided by the client are missing, incomplete or invalid), a `401 - Unauthorized` status response is returned, including a `WWW_AUTHENTICATE` header for each allowed authentication scheme, if any.

3. The property configured `SecurityContext` can be later obtained in the JAX-RS resource (using for example the standard `@Context` annotation) to inspect the authenticated *principal* and perform role-based authorization controls. See Using `AuthenticationInspector` with JAX-RS `SecurityContext` for more advanced authentication inspection and authorization controls.

10.2.1. JAX-RS `Realm` configuration

As stated in the previous section, the JAX-RS `AuthenticationFeature` relies on the core `Realm` API to perform actual authentication and authorization operations. For this reason, a `Realm` instance must be configured and available in JAX-RS server context.

The `Realm` instance must be configured with the appropriate `Authenticator` and `Authorizer` sets, according to the authentication and authorization strategies which have to be supported by the JAX-RS application. Furthermore, one or more `AuthenticationTokenResolver` can be configured to extract the authentication credentials from the incoming JAX-RS request message and obtain a suitable `AuthenticationToken` to be submitted to the Realm authenticators.

The `Realm` instance can be provided in two ways:

1. Using a standard JAX-RS `javax.ws.rs.ext.ContextResolver` instance bound to the `Realm` type and registered in JAX-RS server context.

```
class RealmContextResolver implements ContextResolver<Realm> {

  @Override
  public Realm getContext(Class<?> type) {
    return Realm.builder() //
        .withResolver(AuthenticationToken.httpBasicResolver()) ①
        .withAuthenticator(Account.authenticator(getAccountProvider())) ②
        .withDefaultAuthorizer() ③
        .build();
  }

}
```

① Register a message resolver which extract HTTP *Basic* authentication credentials from the request message and provides a `AccountCredentialsToken` configured with such credentials

② Register an `Account` based authenticator to process the `AccountCredentialsToken` and perform authentication using an `AccountProvider` to obtain account informations

③ Register a default `Authorizer`, which uses authentication *permissions* to perform authorization controls

2. Using the the Holon platform Context architecture to provide the `Realm` instance as a context resource, using the `Realm` class name as *resource key*.

```
Context.get().classLoaderScope() ①
    .map(s -> s.put(Realm.CONTEXT_KEY,
        Realm.builder().withResolver(AuthenticationToken.httpBasicResolver())
            .withAuthenticator(Account.authenticator(getAccountProvider()))
.withDefaultAuthorizer()
            .build()));
```

① Register a configured `Realm` instance in the default *classloader* scope, using the default resource key (the `Realm` class name)

10.2.2. Authentication schemes

When more than one **authentication scheme** is supported by the current `Realm`, allowed authentication schemes for each JAX-RS resource class or method can be specified using the `schemes()` attribute of the `@Authenticate` annotation.

When the authentication scheme is specified, the authentication will be performed using a matching `AuthenticationTokenResolver` by using the scheme name, if available. For this reason, a suitable, scheme-matching AuthenticationTokenResolver must be registered in `Realm` to perform authentication using a specific authentication scheme.

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

10.3. Using `AuthenticationInspector` with JAX-RS `SecurityContext`

When the Authentication feature is used for JAX-RS `SecurityContext` setup, the JaxrsAuthenticationInspector API can be used to obtain the authenticated *principal* as an `Authentication` (the default authenticated principal representation in the Holon platform architecture) and to perform authorization controls using the `Authentication` granted *permissions*.

The `JaxrsAuthenticationInspector` API can be obtained from a `SecurityContext` instance using the `of` builder method and makes available all the methods provided by the standard `AuthenticationInspector` API.

```
@Authenticate
@GET
@Path("name")
@Produces(MediaType.TEXT_PLAIN)
public String getPrincipalName(@javax.ws.rs.core.Context SecurityContext
securityContext) {
  JaxrsAuthenticationInspector inspector = JaxrsAuthenticationInspector.of
(securityContext); ①

  boolean isAuthenticated = inspector.isAuthenticated(); ②
  Optional<Authentication> auth = inspector.getAuthentication(); ③
  Authentication authc = inspector.requireAuthentication(); ④

  boolean permitted = inspector.isPermitted("ROLE1"); ⑤
  permitted = inspector.isPermittedAny("ROLE1", "ROLE2"); ⑥

  return inspector.getAuthentication().map(a -> a.getName()).orElse(null);
}
```

① Obtain a `JaxrsAuthenticationInspector` from current `SecurityContext`

② Check if an authenticated principal is available

③ Get the `Authentication` reference if available

④ Requires the `Authentication` reference, throwing an exception if the context is not authenticated

⑤ Checks if the role named `ROLE1` is granted to the authenticated principal

⑥ Checks if the role named `ROLE1` or the role named `ROLE2` is granted to the authenticated principal

10.4. 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 stardard `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`.

The role-based authorization control, when the Authentication feature is enabled and the JAX-RS resource class or method is secured using the `@Authenticate` annotation, is performed using the `AuthContext` type `SecurityContext`, that is, is delegated to the *authorizers* registered in the `Realm` which is bound to the authentication context.

This means that, by default, the current `Authentication` *permissions* are used to perform the authorization controls, using the permission's String representation when a role-based authorization control is performed.

See the Authorizer section of the Realm documentation for more information about permissions representation and authorization control strategies.

10.4.1. 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
        .withResolver(AuthenticationToken.httpBasicResolver()) ⑦
        .withAuthenticator(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`)

See the GitHub Holon Platform Examples repository for more examples about JAX-RS authentication and authorization, including examples on how to use **JWT** (JSON Web Tokens) for JAX-RS endpoint authentication.

10.5. JAX-RS HttpRequest

The JaxrsHttpRequest interface represents a `HttpRequest` backed by a JAX-RS request, and can be used as an adapter to obtain a JAX-RS request messages as an `HttpRequest` API, the default Holon platform representation of an HTTP request message.

To create a `HttpRequest` from a JAX-RS request context, the `create(…)` static methods can be used. The creation methods use JAX-RS injectable request information to obtain the concrete request attributes and configuration, such as `Request`, `UriInfo` and `HttpHeaders`.

```
@GET
@Path("name")
@Produces(MediaType.TEXT_PLAIN)
public String ping(@Context Request request, @Context UriInfo uriInfo, @Context
HttpHeaders headers) {

  JaxrsHttpRequest req = JaxrsHttpRequest.create(request, uriInfo, headers); ①

  Optional<Locale> locale = req.getLocale(); ②

  return "pong";
}
```

① Build a `JaxrsHttpRequest` from current request information

② Get the request language, if available

See the MessageAuthenticator documentation for information about *message authenticators* and to learn how to use the `HttpRequest` API to perform message-based authentication.

11. Spring Security integration

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-security</artifactId>
<version>5.5.0</version>
```

The SpringSecurityAuthenticationFeature JAX-RS feature can be used to enable the `@Authenticate` annotation based authentication strategy using a configured **Spring Security** context as authentication handler.

When this feature is registered and enabled, the authentication strategy and behaviour put in place by the standard Authentication feature can be seamlessy implemented using a **Spring Security** context, instead of a `Realm` based authentication setup.

Just like the standard Authentication feature, when the `@Authenticate` annotation is detected on a JAX-RS endpoint resource class and/or method, the standard JAX-RS `SecurityContext` of the request is replaced with an AuthContext API compatible implementation, which is backed by the concrete Spring Security `SecurityContext`.

This way, the incoming request authentication and authorization is delegated to the Spring Security context, and the possible authenticated *principal* is mapped to a default Holon Platform `Authentication` reference, which can be seamlessy used by the Holon Platform authentication and authorization features and APIs.

See the core Spring Security integration documentation for details about the integration between

the Holon Platform authentication/authorization architecture and the Spring Security one.

11.1. Feature configuration

When the `holon-jaxrs-spring-security` artifact is present in classpath, the `SpringSecurityAuthenticationFeature` is automatically registered for **Jersey** and **Resteasy** JAX-RS server runtimes, leveraging on Jersey *AutoDiscoverable* and Resteasy Java Service extensions features.

Just like the standard Authentication feature, the `holon.jaxrs.server.disable-authentication` property name can be used to explicitly disable this feature, registering it as a JAX-RS server configuration property name (with an arbitrary not null value).

12. Spring Boot integration

The JAX-RS module **Spring Boot** integration provides auto-configuration facilities to:

- Auto-configure a JAX-RS `RestClient` implementation.
- Auto-configure the Authentication feature when a `Realm` type bean is detected.
- Simplify the Jersey auto-configuration.
- Enable the Resteasy auto-configuration.

Futhermore, a set of Spring Boot starters are available to provide a quick JAX-RS server and/or client application setup using the Maven dependency system.

12.1. JAX-RS Client

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-client</artifactId>
<version>5.5.0</version>
```

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

Besides the `JaxrsClientBuilder` bean type, a `RestClient` factory is automatically registered, allowing to simply obtain a `RestClient` API instance through the `RestClient.create()` static method, using the `javax.ws.rs.client.Client` instance provided by the `JaxrsClientBuilder` API.

See JAX-RS `RestClient` implementation for more information about the JAX-RS `RestClient` API.

To customize the JAX-RS `ClientBuilder` used to obtain the concrete `javax.ws.rs.client.Client`

instances, the JaxrsClientCustomizer interface can be used.

Any Spring context 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 JAX-RS `ClientBuilder` instance.

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` (and a `RestClient` API backed by the 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`

12.2. Jersey

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-jersey</artifactId>
<version>5.5.0</version>
```

This artifact provides Jersey Spring Boot auto-configuration classes to simplify the Jersey JAX-RS runtime configuration, the JAX-RS server resources registration and the authentication and authorization features setup.

12.2.1. Automatic JAX-RS server resources registration

When the `holon-jaxrs-spring-boot-jersey` is available in classpath, any Spring context **bean** annotated with the `@Path` or `@Provider` JAX-RS annotations is automatically registered as a JAX-RS server resource, using the default `ResourceConfig` Jersey configuration bean.

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

Furthermore, a default Jersey `ResourceConfig` type bean is created when no other `ResourceConfig` type bean is available in the Spring context.

To disable the automatic JAX-RS bean resources registration, the `holon.jersey.bean-scan` Spring boot application property can be used: when setted to `false`, this auto-configuration feature will be disabled.

12.2.2. Handling the `jersey.config.servlet.filter.forwardOn404` configuration property

When Jersey is registered as a Servlet *filter*, the Spring Boot application configuration property `holon.jersey.forwardOn404` is available to set the (boolean) value of the standard

`jersey.config.servlet.filter.forwardOn404` configuration property.

When setted to `true`, it configures the Jersey filter in order to forward the requests for URLs it doesn't know, instead of responding with a `404` error code.

This can be useful when the Jersey filter is mapped to the root context path but other servlets are mapped to a sub path.

12.2.3. Authentication and authorization

When a `Realm` type bean is detected in Spring context, the JAX-RS server is automatically configured to support **authentication**, registering the Authentication feature (and so enabling the `@Authenticate` annotation detection), and **authorization**, relying on standard `javax.annotation.security.*` annotations.

The auto-configuration class perfoms the following operations:

- Registers a `ContextResolver` providing the `Realm` bean instance.
- Registers the Authentication feature.
- Registers the default Jersey RolesAllowedDynamicFeature to support `javax.annotation.security.*` annotations based authorization.

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

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

12.3. Resteasy

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-resteasy</artifactId>
<version>5.5.0</version>
```

This artifact provides Resteasy Spring Boot auto-configuration classes to **automatically setup a Resteasy JAX-RS server runtime** and configure it using Spring Boot application configuration properties.

Furthermore, it provide auto-configuration classes to simplify the JAX-RS server resources registration and the authentication and authorization features setup.

12.3.1. 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.

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

12.3.2. Resteasy configuration customization

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.

12.3.3. Automatic JAX-RS server resources registration

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

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

12.3.4. Resteasy configuration properties

The ResteasyConfigurationProperties lists the configuration properties (with the `holon.resteasy` prefix) which can be used to setup the Resteasy auto-configuration, using standard Spring Boot configuration property sources.

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

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

12.3.5. Authentication and authorization

When a `Realm` type bean is detected in Spring context, the JAX-RS server is automtically configured to support **authentication**, registering the Authentication feature (and so enabling the `@Authenticate` annotation detection), and **authorization**, relying on standard `javax.annotation.security.*` annotations.

The auto-configuration class perfoms the following operations:

- Registers a `ContextResolver` providing the `Realm` bean instance.
- Registers the Authentication feature.
- Set the `resteasy.role.based.security` context init parameter to `true` to enable `javax.annotation.security.*` annotations based authorization.

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

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

12.4. 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>5.5.0</version>
```

12.4.1. JAX-RS client

Artifact id	Description
`holon-starter-jersey-client`	JAX-RS *client* starter using **Jersey** and **Jackson** as JSON provider
`holon-starter-jersey-client-gson`	JAX-RS *client* starter using **Jersey** and **Gson** as JSON provider

Artifact id	Description
holon-starter-resteasy-client	JAX-RS *client* starter using **Resteasy** and **Jackson** as JSON provider
holon-starter-resteasy-client-gson	JAX-RS *client* starter using **Resteasy** and **Gson** as JSON provider

12.4.2. JAX-RS server

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

13. Swagger / OpenAPI integration

Swagger V2 Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-swagger-v2</artifactId>
<version>5.5.0</version>
```

Swagger/OpenAPI V3 Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-swagger-v3</artifactId>
<version>5.5.0</version>
```

The `holon-jaxrs-swagger-v2` and `holon-jaxrs-swagger-v3` artifacts provide integration with the Swagger *OpenAPI Specification*, for the **Swagger Version 2** and the **Swagger/OpenAPI Version 3** specifications respectively.

The Holon Platform 5.1.x `holon-jaxrs-swagger` artifact is **deprecated** and should be replaced with the `holon-jaxrs-swagger-v2` artifact. The `holon-jaxrs-swagger` is still available in Holon Platform version 5.2.x, acting as an alias only for backward compatibility purposes.

The integration facilities concerns the following aspects:

- A complete integration with the Holon Platform **Property model** data types, specifically with the PropertyBox type, ensuring a consistent representation and serialization using the Swagger/OpenAPI definition models. See `PropertyBox` data type support.
- A convenient `ApiReader` API to obtain a Swagger/OpenAPI definition from a set of API resource classes. See Using the `ApiReader` to generate the API definition model.
- A complete and highly configurable **Spring Boot** integration, with auto-configuration facilities for both **Jersey** and **Resteasy** JAX-RS runtimes. See Swagger Spring Boot integration and auto configuration.

The Swagger/OpenAPI integration architecture is designed to be highly consistent between the Swagger/OpenAPI versions 2 and 3, so in most cases it is enough to put the `holon-jaxrs-swagger-v2` or the `holon-jaxrs-swagger-v3` artifact in classpath to enable the integration features. Or even both: see Enabling Swagger V2 and V3 API endpoints simultaneously.

13.1. `PropertyBox` data type support

When the `holon-jaxrs-swagger-v2` or the `holon-jaxrs-swagger-v3` artifact is in classpath, the Swagger/OpenAPI reader engine is automatically configured to support the `PropertyBox` type as API operations parameter or return type.

If you programmatically use the Swagger/OpenAPI reader or context configuration classes to generate the API definition model, you have to include a proper `ReaderListener` class in the API resource classes to ensure a consistent `PropertyBox` type handling. This class is available from the `CONTEXT_READER_LISTENER` static attribute of the SwaggerV2 interface for the Swagger V2 integration and of the SwaggerV3 interface for the Swagger/OpenAPI V3 integration.

When the Spring Boot integration is enabled or an ApiReader is used, the Swagger/OpenAPI engine is automatically configured for a consistent PropertyBox type support.

13.1.1. PropertyBox property set declaration

A PropertyBox type API parameter or return type is translated in a regular Swagger/OpenAPI *object* type Model/Schema definition, listing all the properties of the PropertyBox **property set** as object attributes.

By default, only the Path type properties are included in the API definition *models* /*schemas*, using the Path **name** as model/schema attribute name and the property **type** as model/schema attribute type.

Since a PropertyBox type class does not provide a fixed PropertySet decalration, the PropertySetRef **annotation has to be used** to declare the PropertySet to use to translate the PropertyBox type into an API model/schema definition. The annotation has to be placed on the PropertyBox type method parameters and/or method return types.

The PropertySetRef annotation **value** can be either: * A class which contains the PropertySet instance as a public static field. * A PropertySet type class: in this case, a new instance of such class will be used.

When the first option is used and more than one PropertySet type static field is available in the provided class, the **field** annotation attribute can be used to specify the field name to use.

For example, given the following SubjectModel interface, which contains a data model declaration for a simple subject entity:

```
public interface SubjectModel {

  static final NumericProperty<Integer> ID = NumericProperty.integerType("id");
  static final StringProperty NAME = StringProperty.create("name");

  static final PropertySet<?> SUBJECT = PropertySet.of(ID, NAME); ①

}
```

① PropertySet definition

The SUBJECT field can be used as PropertyBox property set definition in a JAX-RS API endpoint like this:

```
@Path("subjects")
public class Subjects {

  @GET
  @Path("{id}")
  @Produces(MediaType.APPLICATION_JSON)
  public @PropertySetRef(SubjectModel.class) PropertyBox getById(@PathParam("id") int
id) { ①
    return getSubjectById(id);
  }

  @PUT
  @Path("")
  @Consumes(MediaType.APPLICATION_JSON)
  public Response create(@PropertySetRef(SubjectModel.class) PropertyBox subject) { ②
    createSubject(subject);
    return Response.accepted().build();
  }

}
```

① Declare the `PropertySet` to use for the `getById` operation **return type**, using the `PropertySetRef` annotation

② Declare the `PropertySet` to use for the `create` operation **body parameter**, using the `PropertySetRef` annotation

Since the `SUBJECT` field is the unique `PropertySet` type static field of the `SubjectModel` interface, no additional **field** name specification is required in the `PropertySetRef` annotation.

In this example, the generated API definition will look like this (using the YAML format):

Swagger 2 specification:

```yaml
swagger: "2.0"
paths:
  /subjects/{id}:
    get:
      operationId: "getById"
      produces:
      - "application/json"
      parameters:
      - name: "id"
        in: "path"
        required: true
        type: "integer"
        format: "int32"
      responses:
        200:
          description: "successful operation"
          headers: {}
          schema:
            type: "object"
            properties:
              id:
                type: "integer"
                format: "int32"
              name:
                type: "string"
            title: "PropertyBox"
  /subjects:
    put:
      operationId: "create"
      consumes:
      - "application/json"
      parameters:
      - in: "body"
        name: "body"
        required: false
        schema:
          type: "object"
          properties:
            id:
              type: "integer"
              format: "int32"
            name:
              type: "string"
          title: "PropertyBox"
      responses:
        default:
          description: "successful operation"
definitions: {}
```

Swagger/OpenAPI 3 specification:

```yaml
openapi: 3.0.1
paths:
  /subjects/{id}:
    get:
      operationId: getById
      parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
          format: int32
      responses:
        default:
          description: default response
          content:
            application/json:
              schema:
                title: PropertyBox
                type: object
                properties:
                  id:
                    type: integer
                    format: int32
                  name:
                    type: string
  /subjects:
    put:
      operationId: create
      requestBody:
        content:
          application/json:
            schema:
              title: PropertyBox
              type: object
              properties:
                id:
                  type: integer
                  format: int32
                name:
                  type: string
      responses:
        default:
          description: default response
          content:
            '*/*': {}
components:
  schemas: {}
```

See the next section to learn how to include a `PropertyBox` type model/schema in the Swagger API definition `definitions` (for Swagger V2) or `components/schemas` (for Swagger V3).

13.1.2. Creating a Swagger Model/Schema definition for a `PropertyBox` type

When the `PropertySetRef` annotation is used, a generic Swagger *object* type model/schema definition is generated for each API response or API operation parameter, using the property set definition to generate the schema properties.

To include a `PropertyBox` type as a *named* model/schema in the Swagger API definition (i.e. to create a type declaration in the `definitions` section for Swagger V2 or in the `components/schemas` section for Swagger V3), the ApiPropertySetModel annotation can be used.

The `ApiPropertySetModel` **value** attribute is used to specify the model/schema *name*. When a `PropertyBox` type model is declared this way, its name will be used as a *reference* in the API operation components which use it.

The `ApiPropertySetModel` annotation should be always used in conjunction with the `PropertySetRef` annotation, which provides the property set declaration.

To make the API enpoints code more readable and to ensure a consistent property set and model name definition, the `PropertySetRef` and `ApiPropertySetModel` annotations can be used as meta-annotations to create specific annotations to be used in the API endpoints code. See below for an example.

Taking the Subjects example endpoint declaration, we can create a `Subject` annotation to declare the `PropertySet` to use and a model/schema definition named `Subject` this way:

```
@PropertySetRef(SubjectModel.class) ①
@ApiPropertySetModel("Subject") ②
@Target({ ElementType.PARAMETER, ElementType.TYPE, ElementType.TYPE_USE })
@Retention(RetentionPolicy.RUNTIME)
public @interface Subject {

}
```

① `PropertySet` reference declaration

② Model/Schema **name** definition

Next, the `Subject` annotation is used in the JAX-RS endpoint instead of the simple `PropertySetRef` annotation:

```
@Path("subjects")
public class Subjects2 {

  @GET
  @Path("{id}")
  @Produces(MediaType.APPLICATION_JSON)
  public @Subject PropertyBox getById(@PathParam("id") int id) { ①
    return null;
  }

  @PUT
  @Path("")
  @Consumes(MediaType.APPLICATION_JSON)
  public Response create(@Subject PropertyBox subject) { ②
    return Response.accepted().build();
  }

}
```

① Declare the `Subject` model definition for the `getById` operation **return type**

② Declare the `Subject` model definition for the `create` operation **body parameter**

The generated API definition will include a `Subject` model/schema definition and will look like this:

Swagger 2 specification:

```yaml
swagger: "2.0"
paths:
  /subjects/{id}:
    get:
      operationId: "getById"
      produces:
      - "application/json"
      parameters:
      - name: "id"
        in: "path"
        required: true
        type: "integer"
        format: "int32"
      responses:
        200:
          description: "successful operation"
          headers: {}
          schema:
            $ref: "#/definitions/Subject"
  /subjects:
    put:
      operationId: "create"
      consumes:
      - "application/json"
      parameters:
      - in: "body"
        name: "body"
        required: false
        schema:
          $ref: "#/definitions/Subject"
      responses:
        default:
          description: "successful operation"
definitions:
  Subject:
    type: "object"
    properties:
      id:
        type: "integer"
        format: "int32"
      name:
        type: "string"
    title: "Subject"
```

Swagger/OpenAPI 3 specification:

```yaml
openapi: 3.0.1
paths:
  /subjects/{id}:
    get:
      operationId: getById
      parameters:
      - name: id
        in: path
        required: true
        schema:
          type: integer
          format: int32
      responses:
        default:
          description: default response
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Subject'
  /subjects:
    put:
      operationId: create
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Subject'
      responses:
        default:
          description: default response
          content:
            '*/*': {}
components:
  schemas:
    Subject:
      title: Subject
      type: object
      properties:
        id:
          type: integer
          format: int32
        name:
          type: string
```

In the API definition, the API operation return types or body parameters annotated with the `Subject` annotation declarations provide a **reference** to the `Subject` model/schema definition.

13.2. Using the `ApiReader` to generate the API definition model

The ApiReader service can be used to directly generate a Swagger API definition model from a set of JAX-RS API resource classes, ensuring a consistent Swagger engine configuration with the Holon Platform property model support.

The `read(Set<Class<?>> classes)` method reads and generates a Swagger API definition model which includes the given API resource classes operations.

13.2.1. Swagger version 2 `ApiReader`

The SwaggerV2 entrypoint interface can be used to obtain an `ApiReader`, using a `SwaggerConfig` type configuration definition and providing a `io.swagger.models.Swagger` type API definition model.

Furthermore, the `SwaggerV2` interface provides a set of convenience methods to serialize the Swagger model using **JSON** (through the `asJson(…)` methods) or **YAML** (through the `asYaml(…)` methods).

```
BeanConfig configuration = new BeanConfig();
configuration.setTitle("The title");
configuration.setVersion("1");

ApiReader<Swagger> reader = SwaggerV2.reader(configuration); ①

Swagger api = reader.read(ApiEndpoint1.class, ApiEndpoint2.class); ②

String json = SwaggerV2.asJson(api); ③
String yaml = SwaggerV2.asYaml(api); ④
```

① Obtain a `ApiReader` using the provided configuration

② Generate the Swagger API model from given API resource classes

③ Serialize the API model using JSON

④ Serialize the API model using YAML

13.2.2. Swagger/OpenAPI version 3 `ApiReader`

The SwaggerV3 entrypoint interface can be used to obtain an `ApiReader`, using a `OpenAPIConfiguration` type configuration definition and providing a `io.swagger.v3.oas.models.OpenAPI` type API definition model.

Furthermore, the `SwaggerV3` interface provides a set of convenience methods to serialize the Swagger/OpenAPI model using **JSON** (through the `asJson(…)` methods) or **YAML** (through the `asYaml(…)` methods).

```
SwaggerConfiguration configuration = new SwaggerConfiguration();
configuration.setOpenAPI(new OpenAPI().info(new Info().title("The title").version("1"
)));

ApiReader<OpenAPI> reader = SwaggerV3.reader(configuration); ①

OpenAPI api = reader.read(ApiEndpoint1.class, ApiEndpoint2.class); ②

String json = SwaggerV3.asJson(api); ③
String yaml = SwaggerV3.asYaml(api); ④
```

① Obtain a `ApiReader` using the provided configuration

② Generate the OpenAPI model from given API resource classes

③ Serialize the API model using JSON

④ Serialize the API model using YAML

13.3. Swagger Spring Boot integration and auto configuration

The `holon-jaxrs-swagger-v2` and `holon-jaxrs-swagger-v3` artifacts provide a set of **Spring Boot** auto-configuration classes to automatically configure **Swagger API definition JAX-RS endpoints** for Swagger/OpenAPI version 2 and version 3 respectively, using the Spring Boot application JAX-RS API endpoints classes as API definition sources.

Both **Jersey** and **Resteasy** JAX-RS runtimes are supported.

Obviously, all the standard **Swagger annotations** are supported for the API definition configuration. See Swagger specification version 2 annotations and Swagger/OpenAPI specification version 3 annotations for information about the available annotations.

13.3.1. Prerequisites

To enable the Swagger Spring Boot auto-configuration facilities, just include in classpath the `holon-jaxrs-swagger-v2` artifact for Swagger version 2 integration and `holon-jaxrs-swagger-v3` artifact for Swagger/OpenAPI version 3 integration.

It is perfectly safe to include **both** `holon-jaxrs-swagger-v2` and `holon-jaxrs-swagger-v3` artifacts in order to enable the Swagger version 2 and version 3 integration simultaneously. See Enabling Swagger V2 and V3 API endpoints simultaneously for JAX-RS API definition endpoints *path* naming considerations.

The Swagger API definition endpoints auto-configuration is triggered when the following conditions are satisfied:

- When the **Jersey** JAX-RS runtime is used, a `org.glassfish.jersey.server.ResourceConfig` type bean must be available in context.

- When the **Resteasy** JAX-RS runtime is used, a `com.holonplatform.jaxrs.spring.boot.resteasy.ResteasyConfig` type bean must be available in context.

When the Holon Platform JAX-RS Spring Boot integration is available, a `ResourceConfig` or `ResteasyConfig` type bean can be automatically configured, including the automatic registration of the `@Path` annotated JAX-RS endpoint classes declared as Spring beans. See the JAX-RS Spring Boot integration documentation section.

13.3.2. Default API endpoints configuration strategy

By default, any `@Path` annotated JAX-RS endpoint class registered in the JAX-RS `javax.ws.rs.core.Application` is included in the Swagger API definition.

A JAX-RS **Swagger API definition** endpoint class in generated and registered in the current JAX-RS runtime to provide the Swagger API definition either in **JSON** or **YAML** format.

By default, the Swagger API definition endpoint will be available at the `/api-docs` path and a `type` named URL **query parameter** can be used to declare the API output format, which can be either `json` or `yaml`.

Depending on the `holon-jaxrs-swagger-v2` or `holon-jaxrs-swagger-v3` artifact presence in classpath, the Swagger API definition output will be provided either in version 2 or version 3 specification format.

For example, given a Maven project whith the following dependencies in the `pom.xml` file:

```
<dependencies>
  <!-- Optional Holon JAX-RS starter using Jersey -->
  <dependency>
    <groupId>com.holon-platform.jaxrs</groupId>
    <artifactId>holon-starter-jersey</artifactId>
    <version>5.5.0</version>
  </dependency>
  <!-- Swagger V3 integration -->
  <dependency>
    <groupId>com.holon-platform.jaxrs</groupId>
    <artifactId>holon-jaxrs-swagger-v3</artifactId>
    <version>5.5.0</version>
  </dependency>
</dependencies>
```

A Swagger API definition endpoint will be automatically configured and mapped to the `/api-docs` path. Supposing the application is available from `localhost`, the API definition can be obtained from the following URL:

JSON format:

```
http://localhost/api-docs?type=json
```

YAML format:

```
http://localhost/api-docs?type=yaml
```

The Swagger Spring Boot integration can be widely configured, either using explicit configuration beans or the `holon.swagger.*` configuration property set. See the next sections for the available configuration options.

13.3.3. API definition endpoint path

By default, the Swagger API definition endpoint is mapped to the `/api-docs` path.

See the API and endpoints configuration using beans section or the API and endpoints configuration using application properties section to learn how to specify the Swagger API definition endpoint path, either using the `@ApiConfiguration` bean annotation or the `holon.swagger.path` application configuration property.

13.3.4. API definition endpoint types

The following Swagger API definition JAX-RS endpoint types are available and listed in the ApiEndpointType enumeration:

Type	Description	Parameters
`QUERY_PARAMETER`	Swagger API definition endpoint which uses a `type` named URL **query parameter** for the API output format declaration.	The `type` query parameter valid values are `json` for JSON format and `yaml` for YAML format. If not specified, the JSON format is used by default.
`PATH_PARAMETER`	Swagger API definition endpoint which uses a **path parameter** for the API output format declaration. The path parameter value must be prefixed by a dot (`.`). For example: `http://localhost/api-docs.json` or `http://localhost/api-docs.yaml`.	The path parameter valid values are `json` for JSON format and `yaml` for YAML format. If not specified, the JSON format is used by default.

Type	Description	Parameters
`ACCEPT_HEADER`	Swagger API definition endpoint which uses the `Accept` **request header** value for the API output format declaration.	The `Accept` request header valid values are `application/json` for JSON format and `application/yaml` for YAML format. If not specified, the JSON format is used by default.

The `QUERY_PARAMETER` type is the **default** Swagger API definition endpoint type. See the API and endpoints configuration using beans section or the API and endpoints configuration using application properties section to learn how to specify the Swagger API definition endpoint type, either using the `@ApiConfiguration` bean annotation or the `holon.swagger.type` application configuration property.

13.3.5. API resources scan strategy

The following strategies are available to detect the JAX-RS API resource classes to include in the Swagger API definition (listed in the JaxrsScannerType enumeration):

Type	Strategy
`APPLICATION`	Only the `@Path` annotated resource classes registered in the JAX-RS `Application` will be included in the Swagger API definition.
`ANNOTATION`	All the `@Path` annotated resource classes available from the Spring application scan paths will be included in the Swagger API definition.
`APPLICATION_AND_ANNOTATION`	Both the `@Path` annotated resource classes registered in the JAX-RS `Application` and the `@Path` annotated resource classes available from the Spring application scan paths will be included in the Swagger API definition.

By default, the `APPLICATION` type scan strategy is used. See the API and endpoints configuration using beans section or the API and endpoints configuration using application properties section to learn how to specify the API definition resources scan strategy, either using the `@ApiConfiguration` bean annotation or the `holon.swagger.scanner-type` application configuration property.

13.3.6. API and endpoints configuration using application properties

The `holon.swagger.*` configuration property set can be used to configure the Swagger API definition and the Swagger API endpoints.

The SwaggerConfigurationProperties class represents the Spring Boot application properties which are available for configuration.

1. Swagger auto-configuration enabling:

Property name	Type	Meaning	Default
holon.swagger.**enabled**	Whether the Swagger API definition endpoints auto-configuration is enabled	Boolean	`true`

2. API endpoint configuration:

Property name	Type	Meaning	Default
holon.swagger.**path**	The Swagger API definition endpoint JAX-RS path	String	`api-docs`
holon.swagger.**type**	The Swagger API definition endpoint type. Must be one of the ApiEndpointType enumeration values. See API definition endpoint types.	String	`QUERY_PARAMETER`
holon.swagger.**scanner-type**	The Swagger API definition resource classes scan strategy. Must be one of the JaxrsScannerType enumeration values. See API resources scan strategy.	String	`APPLICATION`
holon.swagger.**context-id**	The context id to which the generated Swagger API definition is bound.	String	`openapi.context.id.default`
holon.swagger.**resource-package**	The package names to use to filter the API resource classes to be included in the Swagger API definition. More than one package name can be specified using a comma (`,`) as separator.	String	*None*

Property name	Type	Meaning	Default
holon.swagger.**include-all**	Whether to include all the available API resource classes in the Swagger API definition. If `false`, only the `io.swagger.annotations.Api` annotated resource classes will be incuded for Swagger V2 or only the `io.swagger.v3.oas.annotations.Operation` annotated resource methods will be included for Swagger V3.	Boolean	`true`
holon.swagger.**ignored-routes**	A comma separated list of API operation routes to be ignored for the Swagger API definition generation.	String	*None*
holon.swagger.**pretty-print**	Whether *pretty* format the Swagger API definition output provided by the API definition endpoints.	Boolean	`false`
holon.swagger.**v2.path**	Specific API definition endpoint JAX-RS path for Swagger API specification version 2. Overrides the default `holon.swagger.path` when the Swagger V2 integration is available from classpath. See Enabling Swagger V2 and V3 API endpoints simultaneously.	String	*None*

Property name	Type	Meaning	Default
holon.swagger.**v3.path**	Specific API definition endpoint JAX-RS path for Swagger/OpenAPI specification version 3. Overrides the default `holon.swagger.path` when the Swagger V3 integration is available from classpath. See Enabling Swagger V2 and V3 API endpoints simultaneously.	String	*None*

3. API definition configuration:

Property name	Type	Meaning	Default
holon.swagger.**title**	The API definition title.	String	*None*
holon.swagger.**version**	The API definition version.	String	*None*
holon.swagger.**description**	The API definition description.	String	*None*
holon.swagger.**terms-of-service-url**	The API definition terms of service URL.	String	*None*
holon.swagger.**contact.name**	The API definition contact name.	String	*None*
holon.swagger.**contact.email**	The API definition contact email.	String	*None*
holon.swagger.**contact.url**	The API definition contact URL.	String	*None*
holon.swagger.**license.name**	The API definition license name.	String	*None*
holon.swagger.**license.url**	The API definition license URL.	String	*None*
holon.swagger.**external-docs.url**	The API definition external documentation URL.	String	*None*
holon.swagger.**external-docs.description**	The API definition external documentation description.	String	*None*

Property name	Type	Meaning	Default
holon.swagger.**server.url**	The API definition server URL.	String	*None*
holon.swagger.**server.description**	The API definition server description.	String	*None*
holon.swagger.**security**	The API security requirements definition. See the example below.	A List of (String, List<String>) pairs	*None*

4. Deprecated API definition properties, from version 5.2.x:

Property name	Meaning	Replaced by
holon.swagger.**license-url**	The API definition license URL.	holon.swagger.license.url
holon.swagger.**schemes**	The API definition server schemes.	holon.swagger.server.url
holon.swagger.**host**	The API definition server host.	holon.swagger.server.url

Example:

application.yml

```
holon:
  swagger:
    title: 'My title'
    version: 'v1'
    path: 'docs'
    type: PATH_PARAMETER
    scanner-type: APPLICATION_AND_ANNOTATION
    pretty-print: true
    security:
      - name: 'requirement 1'
        value: 'value1'
      - name: 'requirement 2'
        value: 'value2,value3'
```

In the example above the Swagger API definition endpoint will be mapped to the `docs` path. The Swagger API definition endpoint will be of `PATH_PARAMETER` type and the API resources scan strategy to use is setted to `APPLICATION_AND_ANNOTATION`.

13.3.7. API and endpoints configuration using beans

Besides the API and endpoints configuration using application properties, the API endpoints and the API definition can be configured using an appropriate configuration class, declared as a **Spring bean** in the application context.

The required configuration bean type depends on the Swagger API specification version:

- For Swagger API specification **version 2**: A `io.swagger.config.SwaggerConfig` bean type is required.
- For Swagger/OpenAPI specification **version 3**: A `io.swagger.v3.oas.integration.api.OpenAPIConfiguration` bean type is required.

When a suitable API configuration bean is available in the application context, it will be used for API definition endpoints auto-configuration and **any** `holon.swagger.` configuration property will be ignored*.

Swagger V2

```
@Component
public class ApiConfigV2 extends BeanConfig {

  public ApiConfigV2() {
    super();
    setTitle("Test bean config");
    setVersion("1.0.0");
  }

}
```

Swagger V3

```
@Component
public class ApiConfigV3 extends SwaggerConfiguration {

  public ApiConfigV3() {
    super();
    setOpenAPI(new OpenAPI().info(new Info().title("Test bean config").version("1.0.0
")));
  }

}
```

For what concern the API definition endpoint, the ApiConfiguration annotation can be used on the configuration bean classes.

The `@ApiConfiguration` annotation allows to configure:

- The API **context id** to which the API definition endpoint is bound, using the `contextId` attribute.
- The JAX-RS **path** to which the API definition endpoint is mapped, using the `path` attribute.
- The API definition **endpoint type**, using the `endpointType` attribute. See API definition endpoint types.
- The **scanner type** to use to detect the API resource classes, using the `scannerType` attribute. See

API resources scan strategy.

Swagger V2 with ApiConfiguration annotation

```
@ApiConfiguration(contextId = "my_context_id", path = "docs", endpointType =
ApiEndpointType.ACCEPT_HEADER, scannerType = JaxrsScannerType
.APPLICATION_AND_ANNOTATION)
@Component
public class ApiConfigV2b extends BeanConfig {

  public ApiConfigV2b() {
    super();
    setTitle("Test bean config");
    setVersion("1.0.0");
  }

}
```

Swagger V3 with ApiConfiguration annotation

```
@ApiConfiguration(contextId = "my_context_id", path = "docs", endpointType =
ApiEndpointType.ACCEPT_HEADER, scannerType = JaxrsScannerType
.APPLICATION_AND_ANNOTATION)
@Component
public class ApiConfigV3b extends SwaggerConfiguration {

  public ApiConfigV3b() {
    super();
    setOpenAPI(new OpenAPI().info(new Info().title("Test bean config").version("1.0.0
")));
  }

}
```

More than one API configuration bean can be declared to register multiple Swagger API definition endpoints, see Multiple API endpoints configuration.

Swagger API specification version 2 and 3 type configuration beans can coexist in the same application context: see Enabling Swagger V2 and V3 API endpoints simultaneously.

13.3.8. Multiple API endpoints configuration

More than one Swagger API definition endpoint can be configured for the same application. This can be typically used to provide an API definition for a **different API resources subset** for each endpoint.

Each API definition endpoint registered in the same application must be mapped to a **different JAX-RS path**.

In this scenario, the default JAX-RS path (`api-docs`) cannot be used for every API definition endpoint when more than one is registered to avoid path conflicts.

The API definition endpoint JAX-RS path can be declared using the `holon.swagger.path` configuration property (see API and endpoints configuration using application properties) or the `path` attribute of the `@ApiConfiguration` annotation, if a bean based configuration is used (see API and endpoints configuration using beans).

See below to learn how to group the API resource classes to create different API definition subsets.

Using a package prefix to group the API resources

When a **package prefix** is used to group the API resources, each API definition group will contain the API resource classes which package name starts with the specified package prefix.

The package prefix can be specified either using a API configuration bean or using the `holon.swagger.api-groups` configuration property.

1. Bean based API configuration:

One or more package prefix can be configured using a Swagger API configuration bean. See API and endpoints configuration using beans for information about the bean class to use for Swagger API definition version 2 or version 3.

The `@ApiConfiguration` annotation can be used on each API configuration bean to specify the API definition endpoint JAX-RS **path**.

Swagger V2 API groups configuration

```
@ApiConfiguration(path = "docs1") ①
@Component
public class ApiConfigV2Group1 extends BeanConfig {

  public ApiConfigV2Group1() {
    super();
    setTitle("API group 1");
    setResourcePackage("my.resource.package.group1");
  }

}

@ApiConfiguration(path = "docs2") ②
@Component
public class ApiConfigV2Group2 extends BeanConfig {

  public ApiConfigV2Group2() {
    super();
    setTitle("API group 2");
    setResourcePackage("my.resource.package.group2");
  }

}
```

① API group 1 configuration: will include the API resource classes which package name starts with the `my.resource.package.group1` package name. The API definition endpoint is mapped to the `docs1` path.

② API group 2 configuration: will include the API resource classes which package name starts with the `my.resource.package.group2` package name. The API definition endpoint is mapped to the `docs2` path.

Swagger V3 API groups configuration

```
@ApiConfiguration(path = "docs1") ①
@Component
public class ApiConfigV3Group1 extends SwaggerConfiguration {

  public ApiConfigV3Group1() {
    super();
    setOpenAPI(new OpenAPI().info(new Info().title("API group 1")));
    setResourcePackages(Collections.singleton("my.resource.package.group1"));
  }

}

@ApiConfiguration(path = "docs2") ②
@Component
public class ApiConfigV3Group2 extends SwaggerConfiguration {

  public ApiConfigV3Group2() {
    super();
    setOpenAPI(new OpenAPI().info(new Info().title("API group 2")));
    setResourcePackages(Collections.singleton("my.resource.package.group2"));
  }

}
```

① API group 1 configuration: will include the API resource classes which package name starts with the `my.resource.package.group1` package name. The API definition endpoint is mapped to the `docs1` path.

② API group 2 configuration: will include the API resource classes which package name starts with the `my.resource.package.group2` package name. The API definition endpoint is mapped to the `docs2` path.

1. Application properties based API configuration:

When the Spring Boot application configuration properties are used for Swagger API definition endpoints configuration, the `holon.swagger.api-groups` configuration property can be used to declare a set of API definition groups.

Each group should provide a **group id** (which will be used as API definition context id) and an optional set of API definition and API endpoint configuration properties. The available configuration properties for each group are the same of the default Swagger configuration property set. See API and endpoints configuration using application properties.

The package prefix to use for each API definition group can be specified using the `resource-package` configuration property. The property allows to specify more than one package prefix for each group, using a comma (`,`) as separator.

For example, given the following configuration:

application.yml

```
holon:
  swagger:
    api-groups:
      - group-id: 'group1'
        title: 'API group 1'
        resource-package: 'my.resource.package.group1'
        path: 'docs1'
      - group-id: 'group2'
        title: 'API group 2'
        resource-package: 'my.resource.package.group2'
        path: 'docs2'
```

Two API definition endpoints will be configured:

- One for the `group1` API definition with title *API group 1*, including the API resource classes which package name starts with the `my.resource.package.group1` package name, mapped to the `docs1` path.
- And one for the `group2` API definition with title *API group 2*, including the API resource classes which package name starts with the `my.resource.package.group2` package name, mapped to the `docs2` path.

The *root level* API configuration properties are **inherited by each API group definition**, if the corresponding property is not explicitly specified in the group property set.

For example, given the following configuration:

application.yml

```
holon:
  swagger:
    version: 'v1'
    api-groups:
      - group-id: 'group1'
        title: 'API group 1'
        resource-package: 'my.resource.package.group1'
        path: 'docs1'
      - group-id: 'group2'
        title: 'API group 2'
        resource-package: 'my.resource.package.group2'
        path: 'docs2'
```

The `version` property value (`v1`) will be inherited by both the group 1 and group 2 API definition.

Using the `ApiContextId` annotation to group the API resources

As an alternative to the API resources package prefix grouping mode, the ApiContextId annotation can be used to group the API resource classes.

The `@ApiContextId` annotation can be used on API resource classes to specify the API definition **context id** to which the API resource class belongs. Each class will be included in the API definition with a matching context id.

The API context id for a API definition group can be declared in the following way:

- If the bean based API configuration mode is used, the context id can be declared for each API configuration bean through the `contextId` attribute of the `@ApiConfiguration` annotation.
- If the application properties based API configuration mode is used, the API context id of each API definition group coincides with the **group id**.

For example, given the following configuration:

application.yml

```
holon:
  swagger:
    api-groups:
      - group-id: 'group1'
        title: 'API group 1'
        path: 'docs1'
      - group-id: 'group2'
        title: 'API group 2'
        path: 'docs2'
```

And the following API resource classes:

```
@ApiContextId("group1")
@Path("resource1")
public class Resource1 {

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

}

@ApiContextId("group2")
@Path("resource2")
public class Resource2 {

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

}
```

The `Resource1` class will be included in the `group1` API definition group (and so available from the Swagger API definition endpoint mapped to the `docs1` path), while the `Resource2` class will be included in the `group2` API definition group (and so available from the Swagger API definition endpoint mapped to the `docs2` path).

13.4. Enabling Swagger V2 and V3 API endpoints simultaneously

If both Swagger API specification version 2 and version 3 endpoints should be registered in the same application, it is perfectly safe to include **both** the `holon-jaxrs-swagger-v2` and `holon-jaxrs-swagger-v3` artifacts in classpath, enabling the Swagger API version 2 and version 3 integration simultaneously.

When using a property based API definition configuration, you have to ensure each Swagger API definition endpoints is mapped to a **different JAX-RS path**, to avoid path conflicts between the Swagger version 2 and version 3 API definition endpoints.

For this purpose the `holon.swagger.v2.path` and the `holon.swagger.v3.path` configuration properties can be used instead of the default `holon.swagger.path` property, to declare the Swagger version 2 and version 3 API definition endpoint path respectively.

For example:

application.yml

```
holon:
  swagger:
    title: 'My title'
    version: '1.0.0'
    v2:
      path: 'docs/v2'
    v3:
      path: 'docs/v3'
```

Similarly, the specific v2/v3 configuration property can be used for API definition groups:

application.yml

```
holon:
  swagger:
    title: 'My title'
    version: '1.0.0'
    api-groups:
      - group-id: 'group1'
        resource-package: 'my.resource.package.group1'
        v2:
          path: 'docs1/v2'
        v3:
          path: 'docs1/v3'
      - group-id: 'group2'
        resource-package: 'my.resource.package.group2'
        v2:
          path: 'docs2/v2'
        v3:
          path: 'docs2/v3'
```

13.5. Disabling the Swagger API endpoints auto-configuration

The Spring Boot Swagger API definition endpoints auto configuration can be disabled in the following ways:

1. Setting the `holon.swagger.enabled` configuration property to `false`:

application.yml

```
holon:
  swagger:
    enabled: false
```

2. Excluding the *Jersey* or *Resteasy* auto configuration class form the Spring Boot auto configuration

classes.

For Swagger API specification **version 2**:

Jersey

```
@EnableAutoConfiguration(exclude={JerseySwaggerV2AutoConfiguration.class})
```

Resteasy

```
@EnableAutoConfiguration(exclude={ResteasySwaggerV2AutoConfiguration.class})
```

For Swagger/OpenAPI specification **version 3**:

Jersey

```
@EnableAutoConfiguration(exclude={JerseySwaggerV3AutoConfiguration.class})
```

Resteasy

```
@EnableAutoConfiguration(exclude={ResteasySwaggerV3AutoConfiguration.class})
```

13.6. `ApiDefinition` annotation deprecation

The `@ApiDefinition` annotation, which can be used until Holon Platform version 5.1.x to group the API resource classes (using the API endpoint path as discriminator) is **deprecated** and should be replaced by the context-based `@ApiContextId` annotation.

See Using the `ApiContextId` annotation to group the API resources.

14. 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

15. System requirements

15.1. 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**.