Holon
PLATFORM

Holon Platform JAX-RS Module - Reference manual

Version 5.1.1

Table of Contents

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>5.1.1</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>5.1.1</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 5.1.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.

4. `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

4.1. 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

4.2. 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

5. JAX-RS `RestClient` implementation

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-client</artifactId>
<version>5.1.1</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.

5.1. 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

6. JAX-RS Server

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-server</artifactId>
<version>5.1.1</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.

6.1. 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).

6.2. 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.

6.2.1. 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() //
        .resolver(AuthenticationToken.httpBasicResolver()) ①
        .authenticator(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().resolver(AuthenticationToken
.httpBasicResolver())
        .authenticator(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)

6.2.2. 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`.

6.3. 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

6.4. 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.

6.4.1. 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
      .resolver(AuthenticationToken.httpBasicResolver()) ⑦
      .authenticator(Account.authenticator(provider)) ⑧
      .withDefaultAuthorizer().build();

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

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

  register(realmContextResolver); ⑩
}
```

① JAX-RS endpoint resource protected using `@Authenticate` and `Basic` HTTP authentication scheme

② JAX-RS endpoint resource with only one protected method

③ This method is not protected

④ Only this method of the resource is protected using `@Authenticate` and `Basic` HTTP authentication scheme

⑤ `AccountProvider` to provide available `Account` s to the Realm

⑥ Build a `Realm` to be used for resource access authentication

⑦ Add a *resolver* for HTTP `Basic` scheme authentication messages

⑧ Set the realm *authenticator* using the previously defined `AccountProvider`

⑨ Create a JAX-RS `ContextResolver` to provide the `Realm` instance to use

⑩ Register the Realm `ContextResolver` in JAX-RS application (for example, using a Jersey `ResourceConfig`)

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.

6.5. 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.

7. Spring Security integration

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-security</artifactId>
<version>5.1.1</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.

7.1. 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).

8. 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.

8.1. JAX-RS Client

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-client</artifactId>
<version>5.1.1</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`

8.2. Jersey

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-jersey</artifactId>
<version>5.1.1</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.

8.2.1. 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.

8.2.2. 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.

8.2.3. 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})
```

8.3. Resteasy

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-spring-boot-resteasy</artifactId>
<version>5.1.1</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.

8.3.1. 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.

8.3.2. 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.

8.3.3. 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.

8.3.4. 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

8.3.5. 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})
```

8.4. 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>5.1.1</version>
```

8.4.1. 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

8.4.2. 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

9. Swagger integration

Maven coordinates:

```
<groupId>com.holon-platform.jaxrs</groupId>
<artifactId>holon-jaxrs-swagger</artifactId>
<version>5.1.1</version>
```

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

Furthermore, a set of **Spring Boot** auto-configuration classes is provided for Swagger API documentation endpoints auto configuration.

9.1. `PropertyBox` support

When the `holon-jaxrs-swagger` artifact is in classpath, the Swagger JAX-RS engine is automatically configured to support the `PropertyBox` type as API parameter or return type.

But to ensure a proper `PropertyBox` type handling in the Swagger API documentation parser, the internal *SwaggerContext* must be setted up. This can be done either by:

- Using the SwaggerConfiguration Swagger `BeanConfig` extension class, instead of the standard `BeanConfig` class.
- Or providing the SwaggerContextListener Swagger `ReaderListener` type class as a Swagger scanned resource class.

When the Spring Boot integration is enabled, the Swagger JAX-RS engine is automatically configured by the module auto-configuration classes.

9.2. `PropertyBox` model definition

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

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

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

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

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

9.3. Spring Boot integration

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

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

9.3.1. Swagger API documentation endpoints configuration using properties

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

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

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

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

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

For example, for a configuration like the following:

application.yml

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

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

Configure multiple API listing endpoints

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

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

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

For example, for a configuration like the following:

application.yml

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

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

Two API groups are defined:

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

9.3.2. Swagger configuration properties

Table 1. Common configuration properties

Name	Meaning
holon.swagger. **enabled**	Whether the Swagger API listing endpoints auto-configuration is enabled. Default is `true`.
holon.swagger. **resourcePackage**	The package name to scan to detect API endpoints
holon.swagger. **path**	API listing endpoint path. When at group level, is appended to the base API listing path.
holon.swagger. **schemes**	API supported protocol schemes list (`http`, `https`)
holon.swagger. **title**	API title
holon.swagger. **version**	API version
holon.swagger. **description**	API description
holon.swagger. **termsOfServiceUrl**	Terms of Service URL
holon.swagger. **contact**	Contact information
holon.swagger. **license**	License information
holon.swagger. **licenseUrl**	License URL
holon.swagger. **host**	API host name

Name	Meaning
holon.swagger. **pretty-print**	Whether to *pretty* format API listing output (`true` or `false`)
holon.swagger. **auth-schemes**	Enable authentication for the API listing endpoints using the `@Authenticate` annotation behaviour, specifying the allowed authentication schemes. If only one scheme with the `*` value is provided, any supported authentication scheme is allowed for authentication.
holon.swagger. **security-roles**	A list of security roles for API listing access control using the JAX-RS `SecurityContext` and the `@RolesAllowed` annotation
holon.swagger. **api-groups**	Optional API groups. Each group can be configured using the properties listed below.

Table 2. API group configuration properties

Name	Meaning
holon.swagger.api-groups. **group-id**	API group id, also used as API listing endpoint sub-path is group `path` is not specified
holon.swagger.api-groups. **resourcePackage**	The package name to scan to detect API group endpoints
holon.swagger.api-groups. **path**	API group listing endpoint path
holon.swagger.api-groups. **schemes**	API group supported protocol schemes list (`http`, `https`)
holon.swagger.api-groups. **title**	API group title
holon.swagger.api-groups. **version**	API group version
holon.swagger.api-groups. **description**	API group description
holon.swagger.api-groups. **termsOfServiceUrl**	Terms of Service URL
holon.swagger.api-groups. **contact**	Contact information
holon.swagger.api-groups. **license**	License information
holon.swagger.api-groups. **licenseUrl**	License URL
holon.swagger.api-groups. **auth-schemes**	Enable authentication for the API group listing using the `@Authenticate` annotation behaviour, specifying the allowed authentication schemes. If only one scheme with the `*` value is provided, any supported authentication scheme is allowed for authentication.
holon.swagger.api-groups. **security-roles**	A list of security roles for API group listing access control using the JAX-RS `SecurityContext` and the `@RolesAllowed` annotation

9.3.3. Swagger API documentation endpoints auto-configuration

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

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

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

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

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

10. 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

11. System requirements

11.1. 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**.