Holon Platform JPA Module - Reference manual

Table of Contents

1. Introduction . . . 1
   1.1. Sources and contributions . . . 2
2. Obtaining the artifacts . . . 2
   2.1. Using the Platform BOM . . . 3
3. What's new in version 5.2.x . . . 3
4. JPA entity bean post processors . . . 3
   4.1. Post processors registration . . . 4
      4.1.1. Direct registration . . . 4
   4.2. JPA bean **property** post processors . . . 4
      4.2.1. Entity identifiers (`@Id` and `@EmbeddedId` annotations) . . . 4
      4.2.2. `@Enumerated` annotation . . . 4
      4.2.3. `@Temporal` annotation . . . 4
      4.2.4. `@Transient` annotation . . . 5
      4.2.5. `@Column` annotation . . . 5
   4.3. JPA bean **property set** post processors . . . 5
      4.3.1. `@Table` annotation . . . 5
5. Loggers . . . 5
   5.1. JPA entity post processors example . . . 5
6. System requirements . . . 7
   6.1. Java . . . 7
   6.2. JPA API . . . 7

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 **JPA** module provides base support for the *Java Persistence API* (JPA).

It makes available a set of bean *post processors* to deal with JPA **entity** annotations, which can be used to configure bean properties at bean introspection time.



See the JPA Datastore documentation to learn how to use the `Datastore` API to

> manage persistent data through a JPA persistence model in an abstract and implementation independent way.

1.1. Sources and contributions

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

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.jpa</groupId>
<artifactId>holon-jpa-bom</artifactId>
<version>6.0.0</version>
```

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

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.holon-platform.jpa</groupId>
      <artifactId>holon-jpa-bom</artifactId>
      <version>6.0.0</version>
      <type>pom</type>
      <scope>import</scope>
    </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 5.2.x

- Support for JDK 9+ module system using `Automatic-Module-Name`.

4. JPA entity bean post processors

Maven coordinates:

```
<groupId>com.holon-platform.jpa</groupId>
<artifactId>holon-jpa-bean-processors</artifactId>
<version>6.0.0</version>
```

The `holon-jpa-bean-processors` artifact provides a set of Bean property and property set **post processors** which can be used to inspect standard **JPA entity annotations** on Java Beans to configure the bean properties accordingly when using the Holon Platform *Property model* and *Bean instrospection* APIs.



See the Property model documentation for information about the Holon Platform Property model and the Java Beans support documentation for information about the Holon Platform Java Beans support and the integration strategies with the *Property model* architecture.

The Holon Platform **Bean instrospection API** supports two types of *post processors* which can be used to extend and enrich the Bean introspection strategy:

- The BeanPropertyPostProcessor API at **property** level, which allows to attend in the Bean properties introspection strategy, i.e. the process to transform each Bean property into a Holon Platform PathProperty representation, tipically to provide additional property configuration capabilities.
- The BeanPropertySetPostProcessor API at **property set** level, which allows to attend in the BeanPropertySet creation strategy, providing additional configuration capabilities.

See the JPA bean **property** post processors and JPA bean **property set** post processors to learn about the available post processors.

4.1. Post processors registration

The JPA post processors provided by the JPA module relies on the standard *Java service extensions* feature to **auto-register** themself in the BeanIntrospector API.

This way, when the `holon-jpa-bean-processors` artifact is available in classpath, the JPA post processors are automatically registered and made available to the Bean introspection API.



The JPA API classes must be available in classpath to enable the JPA bean post processors.

4.1.1. Direct registration

When the auto registration strategy is not applicable for any reason, the JpaBeanPostProcessor API provides a static method to explicitly perform the JPA post processors registration in a `BeanIntrospector` API instance: `registerPostProcessors(BeanIntrospector beanIntrospector)`.

4.2. JPA bean property post processors

4.2.1. Entity identifiers (`@Id` and `@EmbeddedId` annotations)

This post processor deals with the JPA `@Id` and `@EmbeddedId` annotations to detect the JPA entity *identifier* attributes.

Each Bean property declared as JPA entity identifier is declared as Bean `PropertySet` **identifier property**, using the corresponding `PathProperty` representation.

See the IdentifierProperties documentation for information about the property set identifier properties.

4.2.2. `@Enumerated` annotation

This post processor deals with the JPA `@Enumerated` annotation.

When the `@Enumerated` annotation is detected on a Bean property, the post processor configures a PropertyValueConverter for the corresponding `PathProperty` to automatically handle the data model value conversions to the Java *enumeration* type and back, using the **ordinal** or **String** representation according to the `EnumType` declared in the `@Enumerated` annotation.

This post processor skips the converter setup if a `PropertyValueConverter` is already configured for the property, for example using the `@Converter` annotation. See the Builtin Bean post processors documentation section for details.

4.2.3. `@Temporal` annotation

This post processor deals with the JPA `@Temporal` annotation.

When the `@Temporal` annotation is detected on a Bean property, the `TemporalType` configuration

attribute of the corresponding `PathProperty` is setted accordingly to the declared JPA temporal type value.

See the PropertyConfiguration documentation for information about the property temporal type configuration attribute.

4.2.4. `@Transient` annotation

This post processor deals with the JPA `@Transient` annotation.

When the `@Transient` annotation is detected on a Bean property, that property will be **ignored** by the Bean introspection API and will not be part of the eventual Bean property set.

4.2.5. `@Column` annotation

This post processor deals with the JPA `@Column` annotation.

When the `@Column` annotation is detected on a Bean property or getter method and the `name()` attribute is provided, the **column name** is registered in the corresponding `PathProperty` configuration as **data path** mapping, using the `DataMappable.PATH` configuration property.

See the DataMappable documentation for information about the data path representation.

4.3. JPA bean property set post processors

4.3.1. `@Table` annotation

This post processor deals with the JPA `@Table` entity annotation.

When the `@Table` annotation is detected on a Bean class and the `name()` attribute is provided, the **table name** is registered in the `BeanPropertySet` configuration as **data path** mapping, using the `DataMappable.PATH` configuration property.

See the DataMappable documentation for information about the data path representation.

5. 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 name for the **JPA Bean Processors** module is `com.holonplatform.jpa.processors`.

5.1. JPA entity post processors example

The following example shows how the JPA post processors work during JPA entity bean introspection.

```
@Entity
@Table(name = "mytable")
public class MyEntity { ①

  @Id
  private Long id;

  @Column(name = "txt")
  private String textValue;

  @Enumerated(EnumType.ORDINAL)
  private MyEnum enumeration;

  @Temporal(jakarta.persistence.TemporalType.DATE)
  private Date date;

  @Transient
  private String toIgnore;

  // getters and setters omitted
}

void introspect() {

  BeanPropertySet<MyEntity> propertySet = BeanPropertySet.create(MyEntity.class); ②

  Set<PathProperty<?>> identifiers = propertySet.getIdentifiers(); ③

  Optional<String> dataPath = propertySet.property("textValue").getDataPath(); ④

  Optional<TemporalType> temporalType =
propertySet.property("date").getTemporalType(); ⑤

  boolean ignored = propertySet.contains("toIgnore"); ⑥

}
```

① A sample JPA entity definition: the JPA annotations handled by the post processors are declare at bean class and property level

② Obtain a `BeanPropertySet` from the `MyEntity` JPA entity class using bean introspection API

③ The `PathProperty` with path `id` will be configured as property set *identifier* property, since the `id` bean property is marked with the `@Id` annotation

④ A *data path* with value `txt` is configured for the `PathProperty` with path `textValue`

⑤ the `TemporalType.DATE` is is configured for the `PathProperty` with path `date`

⑥ Since the `toIgnore` bean property is marked with `@Transient`, it is ignored during bean introspection and will not be part of the bean property set

6. System requirements

6.1. Java

The Holon Platform JDBC Datastore module requires Java **8** or higher.

6.2. JPA API

The **JPA API** classes must be present in classpath to enable JPA beans processing. The Holon JPA module is tested against the JPA API version **2.0**, **2.1** and **2.2**.