**How To Write an Entity Processor**

In GSRS, an *Entity Processor* is essentially a defined Java event handler, analogous to a database trigger, to be called before or after certain ORM database operations. It can be used to do extra indexing on database objects, enforce data consistency, generate notifications, etc. Entity Processors are a flexible and database neutral way of doing triggers on data transactions.

***When you might want to use one:***

* You want to specially pre-format some data before it’s stored
* You want to persist a calculated value to the database
* You want to maintain an additional index, data view, or cache that would not fit neatly into the Lucene indexing options (\*\* Note: see *How to Write an Index Value Maker,* a separate document)
* You wish to have special notification or logging on certain object-level database operations
* You wish to do additional fail-safe validation on an object before persisting it
* Any time you might want to write a database trigger

***When you might NOT want to use one:***

* You want to add a simple searchable text field or facet to an existing object (consider using an *IndexValueMaker* instead)

**EntityProcessor Interface**

The **EntityProcessor** interface takes a generic **K** that defines the class being processed. The interface defines the following default methods, any of which can be overwritten by an implementation:

```

default void prePersist(K obj) throws FailProcessingException{};

default void postPersist(K obj) throws FailProcessingException{};

default void preRemove(K obj) throws FailProcessingException{};

default void postRemove(K obj) throws FailProcessingException{};

default void preUpdate(K obj) throws FailProcessingException{};

default void postUpdate(K obj) throws FailProcessingException{};

default void postLoad(K obj) throws FailProcessingException{};  
```

* **pre**Persist will be called *before* an initial save operation (insert) is called for that entity
* **post**Persist will be called *after* an initial save operation (insert) is called for that entity, and the transaction is committed
* **pre**Update will be called *before* an update save operation is called for that entity
* **post**Update will be called *after* an update save operation is called for that entity, and the transaction is committed
* **pre**Remove will be called *before* a remove operation (delete) is called for that entity
* **post**Remove will be called *after* a remove operation (delete) is called for that entity, and the transaction is committed
* **post**Load will be called *after* an entity is fetched from the persistence store

**Example Implementation: Standardize Names to Uppercase**

For example, we may want to standardize names to always be upper-case. To do this, we can use an *EntityProcessor* for the Name object. To do this, we can create a `ToUpperCaseNameProcessor` which implements `EntityProcessor<Name>`.

We will need ***prePersist*** and ***preUpdate***to trigger the standardization, so it doesn’t matter if the name is new, or is an update to an existing name.

```

package ix.ginas.processors.ToUpperCaseNameProcessor;

import ix.core.EntityProcessor;

import ix.ginas.models.v1.Name;

public class ToUpperCaseNameProcessor implements EntityProcessor<Name>{

private void standardizeName(Name n){

n.setName(n.getName().toUpperCase());

}

@Override

public void prePersist(Name name) {

standardizeName(name);

}

@Override

public void preUpdate(Name name) {

standardizeName(name);

}

}

```

This will mutate the name object before the actual save.

**Enabling an Entity Processor**

After writing an entity processor, it must be enabled in the application configuration. To do this, first make sure that it is visible to the classpath, generally by adding a dependency entry to the entity service’s POM file. Next, in the config file for the entity service (typically application.conf), it must be explicitly registered for the given entity type that it’s processing. This is done by adding an entry to the `ix.core.entityprocessors` list. We will assume that the EntityProcessor is in the package “ix.ginas.processors”. For the above example, this can be done by adding these lines to the end of the conf file:

```

gsrs.entityProcessors +={

"class":"ix.ginas.models.v1.Name",

"processor":"ix.ginas.processors.ToUpperCaseNameProcessor"

}

```

**Some Notes:**

* You can register multiple EntityProcessors for the same class, or for different classes.
* When you register an EntityProcessor for a class, it will be registered for all subtypes of that class.
* There is currently no guarantee an execution order of the processors, if there is more than 1 registered for an entity.
* If a **pre** operation throws an Exception, it will fail the intended operation, and stop execution of any future processors.
* If a **post** operation throws an Exception, GSRS will continue executing other configured EntityProcessors.

**Writing Tests**

To test an entity processor, add these items to the test class:

* An EntityProcessorFactory. This makes sure the EntityProcessors configured for an entity are called for the configured events.
  + For example,

@Autowired

private TestEntityProcessorFactory entityProcessorFactory;

* + The test entity processor factory simulates what happens during a normal run of GSRS when the entity processor configuration is loaded.
* Make sure the EntityProcessor is available when the test is run.
  + For example,

@Autowired

private PersistedSubstanceProcessorTestDouble persistedSubstanceProcessorTestDouble;

* Use a BeforeEach test event to ‘wire’ the processor
  + For example,

@BeforeEach

public void addEntityProcessor(){

entityProcessorFactory.setEntityProcessors(persistedSubstanceProcessorTestDouble);

persistedSubstanceProcessorTestDouble.reset();

}

Now, within the individual test methods, you can persist or update an entity and verify that the EntityProcessor’s events have taken place.

See the code within our github repository for a complete example:

<https://github.com/ncats/gsrs-spring-module-substances/blob/master/gsrs-module-substance-example/src/test/java/example/substance/processor/PersistedSubstanceProcessorTest.java>