**How To Write an Entity Processor**

**d**

In G-SRS, an *Entity Processor* is essentially a defined Java trigger, to be called before or after certain ORM database operations. They can be used to do extra indexing on database objects, enforce data consistency, generate notifications, etc. Entity Processors are a flexible and database agnostic 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, coming soon)
* 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**, and **MUST** have a public 0 argument constructor. It 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 (update) is called for that entity
* **post**Update will be called *after* an update save operation (update) 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>{

public ToUpperCaseNameProcessor(){}

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 Entity Processor**

After writing an entity processor, it must be enabled in the application. To do this, first make sure that it is visible to the classpath. Next, in the config file for the application (typically ginas.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:

```

ix.core.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 which are entities.
* 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, it will continue executing other hooks EntityProcessors

**Writing Tests**

**Coming soon**