**How To Write an Index Value Maker**

In G-SRS, an *Index Value Maker* is a procedure for taking an Entity and adding searchable and indexable fields, without changing the underlying data model.

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

* You want a calculated value to be searchable, but not stored on the Entity itself
* You want to have facets for data, but do not what a direct hard-link to that data in the data model
* You want to add a new custom sorter for an existing Entity
* You want to add a new type-ahead field

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

* You want to add a new top-level Entity (you must change the data model, or implement your own components for this)
* You want to add a new field to the JSON or Entity itself (you must change the data model, or implement your own components for this)
* You want an index which can not be reduced to simple text or numeric data (e.g. an Index for searching for nearest-neighbor documents -- you must implement your own components for this)

**IndexValueMaker Interface**

The **IndexValueMaker** interface takes a genetic **T**, and **MUST** have a public 0 argument constructor. It defines the following methods, which must be implemented by an implementation:

```

public void createIndexableValues(T t, Consumer<IndexableValue> consumer);  
```

* createIndexableValues creates IndexableValue objects from a supplied Entity object t, and passes them into the supplied Consumer
* IndexableValue is an interface, which supplies a value to be indexed, as well as a field name, and information on how it should be processed (e.g. whether it should be a sortable column, facet, ranged numeric facet, or suggestible field)

**Example Implementation: Add Search Field and Facet for computed value**

For example, we may want to have some String value computable from a Substance be available as a searchable / facetable field. In this example, we’ll use a simple case where we index the first 4 characters of the substance UUIDs. The example could be expanded to do something like fetch information from another database, flat file, etc.

```

package ix.ginas.indexers;

import java.util.function.Consumer;

import ix.core.search.text.IndexValueMaker;

import ix.core.search.text.IndexableValue;

import ix.ginas.models.v1.Substance;

public class SubstanceUUIDIndexValueMaker implements IndexValueMaker<Substance>{

@Override

public void createIndexableValues(Substance t, Consumer<IndexableValue> consumer) {

String firstFour = t.getUuid().toString().substring(0, 4);

consumer.accept(IndexableValue.simpleFacetStringValue("First Four", firstFour));

}

}

```

IndexableValue, as an interface, has a helper method “simpleFacetStringValue” which can construct a basic IndexableValue that will be stored as searchable, and as a facet. After making this basic IndexableValue, we have the consumer consume it, which will trigger the deeper processing and indexing.

**Enabling the Index Value Maker**

After writing an IndexValueMaker, 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.indexValueMakers` list. For the above example, this can be done by adding these lines to the end of the conf file:

```

ix.core.indexValueMakers+={

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

"processor":"ix.ginas.indexers.SubstanceUUIDIndexValueMaker"

}

```

This adds the index value, but does *not* change the UI to make a new facet appear by default. To do this, you can also add the facets you’ve made to the set of default facets in the conf file by adding these lines:

```

ix.ginas.facets.substance.default+="First Four"

```

**Some Notes:**

* You can register multiple IndexValueMaker for the same class, or for different classes.
* When you register an IndexValueMaker 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 value makers, if there is more than 1 registered for an entity.
* There is no need to make only one value, or even one named field per IndexValueMaker. Those field names can be dynamic, and you can have multiple values for the same field name, which just makes each instance searchable.

**Writing Tests**

**Coming soon**