Creating Generators with Scriptrunner

The following script will create a JQL inserter.

```
package examples.docs.structure

// Structure imports
import com.almworks.jira.structure.api.forest.ForestSpec
import com.almworks.jira.structure.api.forest.action.ForestAction
import com.almworks.jira.structure.api.generator.CoreGeneratorParameters
import com.almworks.jira.structure.api.item.CoreIdentities
import com.almworks.jira.structure.api.permissions.PermissionLevel
import com.almworks.jira.structure.api.StructureComponents

// Scriptrunner imports
import com.onresolve.scriptrunner.runner.customisers.PluginModule
import com.onresolve.scriptrunner.runner.customisers.WithPlugin

// Atlassian import (might be available without the import in some instances, but better safe than sorry)
import com.atlassian.jira.component.ComponentAccessor

// A Hashmap
import java.util.Map

@Grab(group = 'com.almworks.jira.structure', module = 'structure-api', version = '16.9.0')

@WithPlugin("com.almworks.jira.structure")

@PluginModule
StructureComponents structureComponents

def structureManager = structureComponents.getStructureManager()
def forestService = structureComponents.getForestService()
def generatorManager = structureComponents.getGeneratorManager()

// For brevity's sake, we will have a permission variable that we pass when we make an admin level request
def permission = PermissionLevel.valueOf("ADMIN")

// The false flag is optional; it is a variable whether we want to search archived structures too
// we are getting the first (zeroth) element because this method returns a list, be mindful if you have multiple
structures with the same name!
def currentStructure = structureManager.getStructuresByName("structureName", permission, false)[0]

// verification log outputs
log.warn "Structure's name is ${currentStructure.getName()}"
log.warn "Structure's Id is ${currentStructure.getId()}"
log.warn "*"*25

// let's try to create a jql query based inserter
def jqltext = "type=Epic";
def Map<String, Object> jqlparams = new java.util.HashMap()
jqlparams.put(CoreGeneratorParameters.JQL, jqltext)
// We have now added our JQL text to the inserter (creating a different kind of inserter works almost exactly the
same way; just build up the params variable differently)
def jqlinserterId = generatorManager.createGenerator("com.almworks.jira.structure:inserter-jql", jqlparams,
currentStructure.getId())

/*
This is where some of the more interesting magic happens. We have an inserter that is attached to the
currentStructure, but it has not been saved yet.
Under the hood, all structures are entities known as forests, so we need to add the generator to the forest,
which we will do by getting the generator item,
getting the current forest and then calling the ForestAction add-on with the generator.
Be mindful, the three zeros here indicate under, after and before rowIds (so once other generators exist, these
will need to be updated). Triple zeros is the first row.
*/
def generatorItem = CoreIdentities.generator(jqlinserterId); // item to add to forest
def forestSource = forestService.getForestSource(ForestSpec.structure(currentStructure.getId())) //resolving
forest source for structure
forestSource.apply(new ForestAction.Add(generatorItem, 0, 0, 0))
```