Groovy Script Note

Ref: <https://hybrismart.com/2017/11/23/useful-groovy-scripts-part-1/>

### **#1. Execute a Flexible Search query**

*flexibleSearchService = spring.getBean("flexibleSearchService");  
query = "select {pk} from {Category}";  
result = flexibleSearchService.search(query);  
itempk = "";  
for (item in result.getResult()) {  
println item.getName();*  
}

### **#2. Retrieve an item by PK**

*import de.hybris.platform.core.PK;*[*Long*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+long)*examplePK = ...;  
object = modelService.get(new PK(examplePK));  
print object.getPk();  
print ", ";  
println object.getName();*

### **#3. Execute a raw SQL query**

*import de.hybris.platform.servicelayer.search.FlexibleSearchQuery;*

*flexyQuery = new FlexibleSearchQuery("SELECT {c:pk} FROM {Customer AS c} WHERE {c.uid} = ?customerId");*

*flexyQuery.addQueryParameter("customerId", "women@hybris.com");*

*result = flexibleSearchService.search(flexyQuery);*

*customer = result.getResult().get(0);*

*print("Name before update: ");*

*println (customer.getName());*

*customer.setName("Name Test");*

*modelService = spring.getBean("modelService");*

*modelService.save(customer);*

*print("Name after update: ");*

*println (customer.getName());*

### **#4. Modifying objects**

import *de.hybris.platform.servicelayer.search.FlexibleSearchQuery*;  
flexibleSearchService = spring.getBean("flexibleSearchService")  
mimeService = spring.getBean("mimeService")  
modelService = spring.getBean("modelService")  
def findMediasWithoutMime() {  
query = "SELECT {PK} FROM {Media} WHERE {mime} IS NULL")  
flexibleSearchService.search(query).result;  
}  
findMediasWithoutMime().each {  
it.mime = mimeService.getMimeFromFileExtension(it.realfilename)  
modelService.save(it)  
}

### **#5. Removing items**

*flexibleSearchService.search("select {pk} from {product} where ....").result.each {  
modelService.remove(it)  
}*

### **#6. Print all properties of an object**

*result = spring.getBean("flexibleSearchService").search("select {pk} from {Language}")  
// properties  
result.properties.each { println "$it.key -> $it.value" }*

### **#7. Print all methods of an object**

*a = spring.getBean("flexibleSearchService").search("select {pk} from {Language}");  
dumpOut a  
def dumpOut( clz ) {  
clz.metaClass.methods.each { method ->  
println "${method.returnType.name} ${method.name}( ${method.parameterTypes\*.name.join( ', ' )} )"  
}  
}*

### **#8. Show the hybris type tree**

*import java.util.\*;  
flexibleSearch = spring.getBean("flexibleSearchService")  
result = flexibleSearch.search (/select {pk} from {ComposedType}/).getResult()  
Tree tree = new Tree();  
result.each {  
Node node = new Node(it.getCode(), it.getSuperType()?.getCode());  
type = it.getClass().getSimpleName();  
type = type.replace("ComposedTypeModel", "<Composed>");  
type = type.replace("RelationMetaTypeModel", "<Relation>");  
type = type.replace("EnumerationMetaTypeModel", "<ENUM>");  
type = type.replace("TypeModel", "");  
node.setDetails(type);  
tree.getElements().add(node);  
}  
  
for (element in tree.getElements()) {  
node1 = tree.find(element.getValue());  
node2 = tree.find(element.getParentValue());  
if (node1 != null) { node1.setParent(node2); }  
if (node2 != null) { node2.addChild(node1); }  
if (element.getParentValue() == null) { root = node1; }  
}  
  
int level = 0;  
printANode(0, root);  
displaySubTree(tree, level, root);  
  
void printANode(level, Node item) {  
print "."\*level;  
println item.getValue() + "(" + item.getDetails() + ")";  
}  
  
void displaySubTree(Tree tree, int level, Node node)  
{  
List<Node> subItems = node.getChildren();  
for (item in subItems) {  
printANode(level+1, item);  
displaySubTree(tree, level+1, item);  
}  
}  
  
public class Tree  
{  
List<Node> elements;  
public List<Node> getElements() { return elements; }  
public Tree() {  
elements = new [ArrayList](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+arraylist)();  
}  
public void add (Node element) {  
elements.add(element);  
}  
  
public Node find(*[*String*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)*value) {  
for (it in elements) { if (it.getValue() == value) { return it; } }  
}  
}  
  
public class Node  
{  
private Node parent = null;  
private List<Node> children = null;  
private*[*String*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)*details = "";  
private*[*String*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)*value = "";  
private*[*String*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)*parentValue = "";  
  
public Node(*[*String*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)*value,*[*String*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)*parent)  
{  
this.children = new ArrayList<>();  
this.value = value;  
this.parentValue = parent;  
}  
  
public setDetails(*[*String*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)*nodeDetails)  
{  
details = nodeDetails;  
}  
public getDetails() { return details; }  
  
public List<Node> getChildren() {  
return children;  
}  
  
public void addChild(Node child)  
{  
children.add(child);  
child.addParent(this);  
}  
public addParent (Node parentNode)  
{  
  
parent = parentNode;  
}  
public getValue () {  
return value;  
}  
public*[*String*](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+string)*getParentValue() {  
return parentValue;  
}  
public setParent(Node node)  
{  
parent = node;  
}  
}*

### **#9. Print hybris type stats**

*import java.util.\*;*

*import java.lang.\*;*

*import de.hybris.platform.servicelayer.search.FlexibleSearchQuery*

*import de.hybris.platform.servicelayer.search.SearchResult;*

*flexibleSearch = spring.getBean("flexibleSearchService")*

*typeService = spring.getBean("typeService");*

*result = flexibleSearch.search (/select {pk} from {ComposedType}/).getResult()*

*Tree tree = new Tree();*

*result.each {*

*Node node = new Node(it.getCode(), it.getSuperType()?.getCode());*

*type = it.getClass().getSimpleName();*

*isabstract = it.getAbstract() ? "<abstract>":"";*

*isjaloonly = it.getJaloonly() ? "<jaloonly>":"";*

*type = type.replace("ComposedTypeModel", "<Composed>");*

*type = type.replace("RelationMetaTypeModel", "<Relation>");*

*type = type.replace("EnumerationMetaTypeModel", "<ENUM>");*

*type = type.replace("TypeModel", "");*

*node.setDetails(type+isabstract+isjaloonly);*

*tree.getElements().add(node);*

*}*

*for (element in tree.getElements()) {*

*node1 = tree.find(element.getValue());*

*node2 = tree.find(element.getParentValue());*

*if (node1 != null) { node1.setParent(node2); }*

*if (node2 != null) { node2.addChild(node1); }*

*if (element.getParentValue() == null) { root = node1; }*

*}*

*displaySubTree(tree, 0, root, root.getValue());*

*void printANode(level, Node item, String history) {*

*count = "-";*

*if (!item.getNotLeaf() && !item.getDetails().contains("<abstract>")*

*&& !item.getDetails().contains("<jaloonly>")) {*

*count = calculateCount(item.getValue())*

*}*

*println history + "\t" + item.getValue() + "(" + item.getDetails() + ") \t"+count;*

*}*

*void displaySubTree(Tree tree, int level, Node node, String history)*

*{*

*List<Node> subItems = node.getChildren();*

*if (subItems.size() == 0) {*

*printANode(level, node, history);*

*}*

*for (item in subItems) {*

*printANode(level+1, item, history );*

*if (item.getChildren().size() != 0) {*

*displaySubTree(tree, level+1, item, history + "=>" + item.getValue());*

*}*

*}*

*}*

*public class Tree*

*{*

*List<Node> elements;*

*public List<Node> getElements() { return elements; }*

*public Tree() {*

*elements = new ArrayList();*

*}*

*public void add (Node element) {*

*elements.add(element);*

*}*

*public Node find(String value) {*

*for (it in elements) { if (it.getValue() == value) { return it; } }*

*}*

*}*

*public class Node*

*{*

*private Node parent = null;*

*private List<Node> children = null;*

*private String value;*

*private String details = "";*

*private String parentValue = "";*

*private Boolean notLeaf = false;*

*public Node(String value, String parent)*

*{*

*this.children = new ArrayList<>();*

*this.value = value;*

*this.parentValue = parent;*

*}*

*public setNotLeaf(Boolean itIsNotALeaf) {*

*notLeaf = itIsNotALeaf;*

*}*

*public getNotLeaf(Boolean itIsNotALeaf) {*

*return notLeaf;*

*}*

*public List<Node> getChildren() {*

*return children;*

*}*

*public void addChild(Node child)*

*{*

*children.add(child);*

*child.addParent(this);*

*}*

*public addParent (Node parentNode)*

*{*

*parent = parentNode;*

*}*

*public getValue () {*

*return value;*

*}*

*public String getParentValue() {*

*return parentValue;*

*}*

*public setParent(Node node)*

*{*

*parent = node;*

*}*

*public getParent()*

*{*

*return parent;*

*}*

*public setDetails(String nodeDetails)*

*{*

*details = nodeDetails;*

*}*

*public getDetails() { return details; }*

*}*

*String calculateCount(component) {*

*query = "select count({pk}) from {"+component+"}";*

*FlexibleSearchQuery fquery = new FlexibleSearchQuery(query);*

*fquery.setResultClassList(Arrays.asList(String.class));*

*fquery.setCount(1);*

*SearchResult<String> result = flexibleSearch.search(fquery);*

*return (result.getResult()?.get(0));*

*}*