Validator API Tutorial

VALIDATIONS

Submitting New Validations

To submit a new validation you use the method

public IJob addNewJob(SgParameters params, List<Integer> ruleIds)

that can be found in the JobActionsAPI class.

There are two types of validations readily available with the library, OAI Content Validations and OAI Usage Validations. We will examine an example for each case.

OAI Usage Validations are used when we want to verify the correct implementation of the OAI-PMH protocol. For instance, let's say we want to check if a particular repository returns records in batch sizes between 100-500 when executing a ListRecords request.

We first need to create a map with all the metadata needed for the new validation.

```
SgParameters params = new SgParameters();
params.addParam(FieldNames.JOB_GENERAL_USER, user);
params.addParam(FieldNames.JOB_GENERAL_TYPE, "OAI Usage Validation");
params.addParam(FieldNames.JOB_OAIUSAGE_BASEURL, baseUrl);
```

Where user is a String containing the username of the user that submits this validation and baseUrl is the base url of the repository we want to test.

Next, we need a list of the rules that wiil be checked during the validation. We want to check the records' batch size, and such a rule is already implemented in the default database created using script.sql. In fact, if you open the table "rules" you can find this rule with id 21.

So, we proceed with the creation of the rules' list.

```
List<Integer> ruleIds = new ArrayList<Integer>();
ruleIds.add(21);
```

Now we are ready to add a new validation.

```
IJob job1 = jobActionsAPI.addNewJob(params, ruleIds);
```

Before starting the execution of this validation, it is wise to register a listener, by implementing the JobListener interface. The done method executes when the validation was completed successfully, whereas the failed method when an exception was raised. The score

parameter is a number between 0 and 100 that quantifies how successful the validation was.

When you have created a new listener you can add it.

```
job1.addListener(listener);
```

And you are ready to submit the validation for execution.

```
jobRunner.startJob(job1);
```

OAI Content Validations are used when we want to check the metadata returned when posing a GetRecord request to the repository. For instance, let's say we want to make sure that all records inside the set named "driver" have a dc:identifier field.

Again, we need to provide the metadata for the validation.

```
SgPrameters params = new SgParameters();
params.addParam(FieldNames.JOB_GENERAL_USER, user);
params.addParam(FieldNames.JOB_GENERAL_TYPE, "OAI Content
Validation");
params.addParam(FieldNames.JOB_OAICONTENT_BASEURL, baseUrl);
params.addParam(FieldNames.JOB_OAICONTENT_RANDOM, "false");
params.addParam(FieldNames.JOB_OAICONTENT_RECORDS, "0");
params.addParam(FieldNames.JOB_OAICONTENT_SET, "driver");
```

Setting *JOB_OAICONTENT_RANDOM* means that the records will be checked in the order they are retrieved when posing a ListRecords request. If it was true, then the records would be selected randomly from the available ones.

JOB_OAICONTENT_RECORDS is the number of records we want to validate. If set to 0 it means all records returned from the ListRecords request.

Finally, we can set *JOB_OAICONTENT_SET* to retrieve records from a particular set (this field is optional).

We then continue as before to create the rule list, register a listener and submit the validation for execution.

```
List<Integer> ruleIds = new ArrayList<Integer>();
ruleIds.add(1);

IJob job1 = jobActionsAPI.addNewJob(params, ruleIds);

job1.addListener(listener);

jobRunner.startJob(job1);
```

Retrieving Validation Summaries

After a validation has finished, you can retrieve a summary containing details on its execution. To do this, you have to use the method

```
public List<Entry> getJobSummary(int jobId)
```

It returns a list of objects of the Entry class. Each such object contains

```
private String name, description;
```

The name and description of the rule that was applied.

```
private String successes;
```

How many of the records that where checked validated successfully.

```
private int weight;
```

The weight of the rule, ie how much it has on the final score.

```
private List<String> errors;
```

Links to the metadata representations of the records that failed to validate.

```
private int ruleId;
```

The id of the rule that was applied.

```
private boolean hasErrors;
```

True iff at least one of the records didn't validate successfully.

RULES

General Guidelines for adding a new rule

Several rule types are already implemented in the library. You can easily add new rule as instances of one of those types. To do this, you have to use the method

public String addNewRule(SgParameters params)

that can be found in the RuleActionsAPI class.

The needed parameter params is a map of strings. Some of the keys that must be specified are the same for every rule type, while others are specific for each rule type. You can find detailed instructions on what to include in the map for each rule type at the end of this tutorial. Here we continue with a specific example.

Let's suppose we want to add a new rule that checks if a record follows a specific vocabulary in its dc:type field.

The correct rule type to implement this functionality is "Vocabulary".

SgParameters params = new SgParameters();

params.addParam(Constants.RULE_TYPE, "Vocabulary");

This rule checks the metadata returned by a GetRecord request, so the rule will be available for validations of the type "OAI Content Validation".

params.addParam(Constants.RULE_JOBTYPE, "OAI Content Validation");

Since this is an important rule, we want to mark it as mandatory.

params.addParam(Constants.RULE_MANDATORY, "true");

At least one dc:type field must follow this vocabulary. So, the rule is successful if >0 of its dc:type fields follow the vocabulary. Other options would be ={number}, >{number} or a. If a was provided, then all dc:type fields would have to be successfully checked for the rule to be considered successful.

params.addParam(Constants.RULE_SUCCESS, ">0");

We want tis rule to impact the final score in a standard way, so we give it a default weight of 1.

params.addParam(Constants.RULE_WEIGHT, "1");

The field we want to check is the dc:type.

params.addParam(Constants.RULE_PROVIDER_INFORMATION, "dc:type");

And finally, we give a good name and description for this rule.

params.addParam(Constants.RULE_NAME, "at least one dc:type follows publications vocabulary");

params.addParam(Constants.RULE_DESCRIPTION, "At least one dc:type field must use one of the values defined in http://info-uri.info/registry/OAIHandler?verb=GetRecord&metadataPrefix=reg&identifier=info:eu-repo");

We then continue with the keys needed for the specific rule type we choose (vocabulary)., which is a comma separated list of the words we want to check if they exist in the dc:type field.

params.addParam(Constants.RULE_VOCABULARY_WORDS, "thesis, report, article");

And finally we are ready to add the new rule by calling addNewRule.

Now, let's suppose we want to add a rule that checks if a repository supports the oai_dc metadata format.

Since this is not a validation that checks the metadata returned by a GetRecord request, we have to provide this rule in the context of an OAI Usage Validation.

params.addParam(Constants.RULE_JOBTYPE, "OAI Usage Validation");

The request we have to send to the repository to check the metadata formats provided is ListMetadataFormats, and the fields we have to check from the resulting xml are named metadataPrefix.

params.addParam(Constants.RULE_PROVIDER_INFORMATION, " verb=ListMetadataFormats, field=metadataPrefix");

Note that we had to specify both the request verb and the field, whereas in an OAI Content Validation we only provided the field. This is because the verb was by default GetRecord.

Since we want to check if there exists at least one metadataPrefix field with the value "oai_dc" we are going to use a Vocabulary type rule.

params.addParam(Constants.RULE_TYPE, "Vocabulary");

params.addParam(Constants.RULE_SUCCESS, ">0");

params.addParam(Constants.RULE_VOCABULARY_WORDS, "oai_dc");

As you can see, we can use rule types in any kind of validation, as long as we provide the correct provider information.

We proceed by specifying the desired values for the rest of the keys, as shown in the first example and then invoke addNewRule.

Rule Types and their Keys

Field Exists

Checks if the field specified in the provider_information exists, eg if a dc:identifier is present. It is semantically equivalent to a rule of the type “Cardinality” with success set to “>0”.

Needed keys: None

Cardinality

Checks if the number of fields are within a given range. Eg, if a record has exactly one dc:date field.

Needed keys:

RULE_CARDINALITY_LESSTHAN – At least that many fields must exist

RULE_CARDINALITY_GREATERTHAN – At most that many fields must exist

Vocabulary

Checks if the contents of a field are one of the specified words.

Needed keys:

RULE_VOCABULARY_WORDS – A comma separated list of the words to be checked

Regular Expression

Checks if the contents of a field match a certain regular expression.

Needed keys:

RULE_REGULAREXPRESSION_REGEXPR – The regular expression

Valid Url

Checks if the contents of a field are a valid url.

Needed Keys: none

Retrievable Resource

Checks if the contents of a field point to a url that contains a retrievable file of a certain mime type. Rules of this type can be configured to crawl inside web pages and look for retrievable resources inside them, and also follow links that might occur up to a specified depth. Caution: This rule type tends to create rules that are very time consuming and IO heavy.

RULE_RETRIEVERESOURCE_MAXDEPTH – The max depth of links that will be followed if an html page is encountered (eg a field contains a jump-off page that contains a retrievable resource)

RULE_RETRIEVERESOURCE_MIMETYPES – A comma separated list of the mime types of the files that will be retrieved, eg “application/pdf, application/msword”

Not Confused Fields

Checks that two fields have different values. Note that for this particular rule type you do not need to specify a provider_information value, since both fields are specified using the following keys

RULE_NOTCONNFUSEDFIELDS_FIELD1 – the first field, eg dc:source

RULE_NOTCONNFUSEDFIELDS_FIELD2 – the second field, eg dc:identifier

Incremental Record Delivery

Executes a series of ListRecord requests with various from and until parameters. Only usable inside an OAI Usage Validation.

Needed keys: Node

Date Granularity

Checks that the granularity of the dc:date field of some randomly picked records matches the granularity reported in the Identify request. Only usable inside an OAI Usage Validation.

Needed keys: Node

Resumption Token Duration Check

Checks the resumption tokens issued during a ListRecords request have a lifespan of at least 24 hours. Only usable inside an OAI Usage Validation.

Needed keys: Node

Schema Validity Check

Checks the xml representation of the ListRecords request to make sure it follows the correct xml schema. Only usable inside an OAI Usage Validation.

Needed keys: Node