IPTC EXTRA project
D1.2 EXTRA API Design

Version 1.0 - 10 March 2017

This deliverable provides the design and definition of the EXTRA API, which supports the maintenance of rules and the classification of documents. The API definition uses the RAML 1.0 specification[1].

Overview

The design of the EXTRA API is largely based on the user stories that are specified in the IPTC EXTRA requirements document[2]. According to it, there are five types of resource of interest:

- Rules
- Document classifications
- Document and Rule Validations
- Schemas
- Dictionaries
- Relevance Algorithms

Hence, each of these resources is directly mapped to a REST endpoint, resulting in the following six endpoints:

- /rules
- /classifications
- /validations
- /schemas
- /dictionaries
- /relevancealgorithms

For each of those, a set of relevant CRUD operations (Create, Retrieve, Update, Delete) are defined and the necessary parameters and response structures are specified when possible. Resource Creation is implemented using the HTTP POST method, Retrieve is implemented using GET, Update using PUT, and Delete using DELETE. The API is still not fully specified, since several of the specifics, e.g. regarding the rule language and the selected relevance algorithms are not fixed at the moment of writing this set of specifications.
The specification of the API is based on RAML 1.0 in order to leverage automatic scaffolding tools and other helpful utilities, such as for instance the RAML2HTML[3] documentation

[1] https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/
[2] https://iptc.org/download/workstream/extra/IPTC-EXTRA-TechnicalRequirements_v100_2017-01-30.pdf
[3] https://github.com/raml2html/raml2html

generator, which automatically creates online documentation based on the specification (cf. Figure 1). RAML editing is done based on the Anypoint Platform[4].
This document presents the five EXTRA API endpoints and is accompanied by a .raml file containing the API specification (Appendix I), along with an HTML file containing the API web-based documentation (Appendix II).

[4] https://anypoint.mulesoft.com

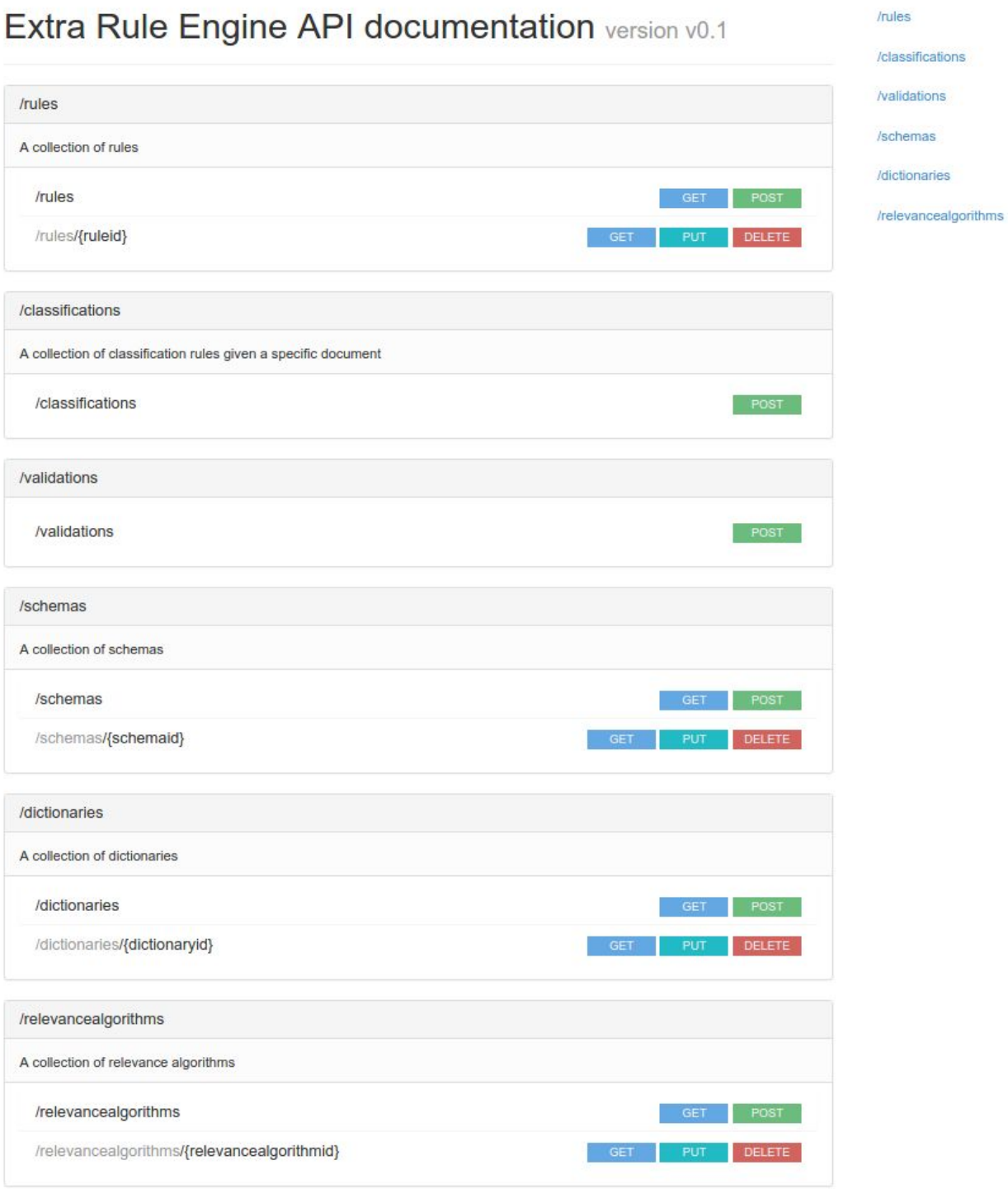

Figure 1: Snapshot from auto-generated HTML documentation based on RAML 1.0 API specification

API Documentation

Note that compared to the actual .raml files that have been delivered, some parts have been omitted for the sake of brevity and cleaner presentation.

<table>
<tr><td colspan="4">#%RAML 1.0
title: Extra Rule Engine API
version: v0.1
mediaType: application/json</td></tr>
<tr><td rowspan="2">types:</td><td>Rule:
type: object
properties:
id:
required: true
type: string
description: The unique identifier of the rule
query:
required: true
type: string
description: The query represent the actual rule, expressed in Extra Rule Language
uid:
required: true
type: string
description: The id of the user that created the rule</td><td>Document:
type: object
properties:
id:
required: true
type: string
schema:
required: false
type: Schema
description: The schema associated with the document</td><td>RelevanceAlgorithm:
type: object
properties:
id:
required: true
type: string
description: A unique identifier of the specific relevance algorithm
name:
required: false
type: string
description: A human-readable name for relevance algorithm
example: FrequencyByWordCount
algorithm:
required: false
type: string
description: A representation of the algorithm in a way that will be defined later
ruleids:
required: false
type: string[]
description: A set of rule ids that this algorithm is a rule-specific algorithm</td></tr>
<tr><td>Schema:
type: object
properties:
name:
required: true
type: string</td><td>Dictionary:
type: object
properties:
id:
required: true
type: string</td><td></td></tr>
</table>

	description: The name of the schema, used as a unique identifier fields: required: true type: string[] description: The set of fields associated with that schema	description: A unique identifier of the dictionary language: required: false type: string description: The language of this dictionary terms: required: true type: string[] description: The set of words appear in this dictionary	

traits	**pageable:** usage: Apply this to any method that needs pagination queryParameters: page: displayName: Page Number type: integer description: Page number of the collection example: 1 default: 1 required: false numPerPage: displayName: Number of items per page type: integer example: 50 default: 20 maximum: 100 required: false	**secured:** usage: Apply this to any method that needs to be secured description: Some requests require authentication. headers: access_token: description: Access Token example: ztVRauPEtguEfWuJnfHDVhWaaS required: true responses: 401: description: This code returned in case of unauthorized access body: application/json: example: | {"message":"Invalid access token"}

Rules

/rules: description: A collection of rules is: secured	
get:	is: pageable description: Get a collection of rules based on filtering criteria like user, status and category queryParameters: uid: displayName: User ID

	type: string
	description: The id of the logged in user. Retrieve rules created by this user.
	example: 1546058f-5a25-4334-85ae-e68f2a44bbaf
	required: true
	status:
	displayName: Rule Status
	type: string
	description: Retrieve rules having the status specified by this parameter.
	example: "submitted"
	required: false
	responses:
	200:
	body:
	application/json:
	example: |
	[
	{
	"id" : "1",
	"uid" : "1546058f-5a25-4334-85ae-e68f2a44bbaf",
	"query" : "title:(donald trump AND us elections) OR (jim mattis AND defense)",
	"status": "editable"
	},
	{
	"id" : "2",
	"uid" : "1546058f-5a25-4334-85ae-e68f2a44bbaf",
	"query" : "title:(us elections) AND body:(barack obama)"
	}
	]
	type: Rule[]

post:	description: Create a new rule as defined in the body of the method
	body:
	application/json:
	type: Rule
	responses:
	201:
	headers:
	Location:
	example: /rules/1
	body:
	application/json:
	type:
	properties:
	message: string
	rule: Rule
	example: |
	{
	"message": "Rule created successfully",
	"rule": {
	"id" : "1",
	"uid" : "1546058f-5a25-4334-85ae-e68f2a44bbaf",
	"query" : "title:(donald trump AND us elections) OR (jim mattis AND defense)"
	}
	}
	400:
	body:
	application/json:
	example: |
	{"message":"Cannot create the rule"}
	409:

```
    body:
      application/json:
        example: |
          {"message":"Conflict. Rule with id=1 already exists"}
```

```
/rules/{ruleid}:
  is: secured
  description: A specific rule, a member of the rules collection
  uriParameters:
    ruleid:
      type: string
```

get:

```
  description: Retrieve the rule defined by the specific ruleid
  responses:
    200:
      body:
        application/json:
          example: |
            {
              "id" : "1",
              "uid" : "1546058f-5a25-4334-85ae-e68f2a44bbaf",
              "query" : "title:(barack obama)",
              }
          type: Rule
    404:
      body:
        application/json:
          type:
            properties:
              message: string
          example: |
            {
              "message": "Rule not found"
            }
```

put:

```
  description: Update the rule defined by the specific ruleid or insert if the rule does not exist
  body:
    application/json:
      type: Rule
  responses:
    201:
      body:
        application/json:
          type:
            properties:
              message: string
              rule: Rule
          example: |
            {
              "message": "Rule updated successfully",
              "rule": {
                "id" : "1",
                "uid" : "1546058f-5a25-4334-85ae-e68f2a44bbaf",
                "query" : "title:(barack obama)"
              }
            }
    400:
      body:
        application/json:
          example: |
```

	{"message": "Rule failed to be updated"} 404: body: application/json: example: | {"message": "Rule not found"}
delete:	description: Delete the rule defined by ruleid={ruleid} responses: 204: body: application/json: type: properties: message: string example: | {"message":"Rule deleted"} 404: body: application/json: type: properties: message: string example: | {"message": "Rule {ruleid} not found"}

Classifications

/classifications: description: A collection of classification rules given a specific document is: secured	

post:	body:
	application/json:
	type:
	properties:
	document: Document
	matches:
	type: object[]
	description: A set of rule IDs for classification
	required: false
	parameters:
	type: object
	description: A set of rule modification parameters
	required: false
	example: |
	{
	"document": {
	"id" : "g1DWjQm2MXFqzdfWr8ka",
	"title" : "this is a test document",
	"body" : "this is the text body of the document"
	},
	"matches": [
	{"ruleid": "1"},

	{"ruleid": "1234"}
	],
	"parameters": {
	"minimum_occurrence": ".25",
	"relevance algorithm": "relalg123",
	"highlight": false
	}
	}
	responses:
	201:
	body:
	application/json:
	type:
	properties:
	found: integer
	matches: object[]
	example: |
	{
	"found": 2,
	"matches": [
	{
	"ruleid" : "1234",
	"relevance": 0.9
	},
	{
	"ruleid" : "456",
	"relevance": 0.75
	}
	]
	}
	400:
	body:
	application/json:
	example: |
	{"message":"Classification failed."}

Validations

/validations: is: secured	
post:	description: Validate a rule or document against a schema (if not specified validation is performed against all body: application/json: type: properties: rule: type: Rule required: false document: type: Document required: false schemas:

	type: string[]
	required: false
	responses:
	200:
	body:
	application/json:
	type:
	properties:
	valid: boolean
	schemas: string[]
	invalidFields: string[]
	examples:
	valid:
	{
	"valid": true,
	"schemas": [
	"1", "45", "109"
	],
	"invalidFields": []
	}
	invalid:
	{
	"valid": false,
	"schemas": [],
	"invalidFields": [
	"_Headline", "_tmac"
	]
	}
	400:
	body:
	application/json:
	type:
	properties:
	message: string
	example: |
	{"message":"Invalid input. Exactly one rule or one document must be specified."}

Schemas

/schemas: description: A collection of schemas is: secured	
get:	description: Get a list of schemas is: pageable responses: 200: body: application/json: type: Schema[]
post:	description: Create a new schema

	body:
	  application/json:
	  type: Schema
	responses:
	  201:
	  body:
	  application/json:
	  type: Schema
	  400:
	  body:
	  application/json:
	  example: |
	  {"message":"Schema failed to be saved"}
	  409:
	  body:
	  application/json:
	  example: |
	  {"message":"Conflict. Schema already exists"}

/schemas:/{schemaid}: is: secured uriParameters: schemaid: type: string	

get:	description: Get a schema having schemaid={schemaid}
	responses:
	  200:
	  body:
	  application/json:
	  type: Schema
	  404:
	  body:
	  application/json:
	  type:
	  properties:
	  message: string
	  example: |
	  {"message" : "Schema not found"}

put:	description: Update schema having schemaid={schemaid}
	body:
	  application/json:
	  type: Schema
	responses:
	  200:
	  body:
	  application/json:
	  type:
	  properties:

	message: string schema: Schema 404: body: application/json: type: properties: message: string example: | {"message" : "Schema not found"}

delete:	description: Delete a schema having schemaid={schemaid}
	responses:
	204:
	body:
	application/json:
	type:
	properties:
	message: string
	example: |
	{
	"message" : "Schema deleted"
	}
	404:
	body:
	application/json:
	type:
	properties:
	message: string
	example: |
	{
	"message" : "Schema not found"
	}

Dictionaries

/dictionaries: description: A collection of dictionaries is: secured	
get:	is: pageable description: Get a collection of dictionaries queryParameters: language: type: string

	required: false description: Filter dictionaries based on their language example: en responses: 200: body: application/json: type: Dictionary[]
post:	body: application/json: type: Dictionary responses: 201: body: application/json: type: Dictionary
/dictionaries/{dictionaryid}:	
get:	description: Get a dictionary having 'dictionaryid={dictionaryid}' responses: 200: body: application/json: type: Dictionary 404: body: application/json: type: properties: message: string example: | {"message":"Dictionary not found"}
put:	body: application/json: type: Dictionary responses: 200: body: application/json: type: properties: message: string dictionary: Dictionary 404: body: application/json:

	type: properties: message: string example: | {"message":"Dictionary not found"}

delete:	description: Delete a dictionary having the specific dictionary
	responses:
	204:
	body:
	application/json:
	type:
	properties:
	message: string
	example: |
	{
	"message" : "Dictionary deleted"
	}
	404:
	body:
	application/json:
	type:
	properties:
	message: string
	example: |
	{
	"message" : "Dictionary not found"
	}

Relevance Algorithms

/relevancealgorithms: description: A collection of relevance algorithms is: secured	
get:	is: pageable description: Get a collection of relevance algorithms responses: 200: body: application/json: type: RelevanceAlgorithm[]
post:	body: type: RelevanceAlgorithm responses:

	201: body: application/json: type: RelevanceAlgorithm
/relevancealgorithms/{relevancealgorithmid}: is: secured	

get:	description: Get relevance algorithm having relevancealgorithmid={relevancealgorithmid}
	responses:
	200:
	body:
	application/json:
	type: RelevanceAlgorithm
	404:
	body:
	application/json:
	type:
	properties:
	message: string
	example: |
	{
	"message" : "Relevance algorithm not found"
	}

put:	description: Update relevance algorithm having relevancealgorithmid={relevancealgorithmid}
	responses:
	200:
	body:
	application/json:
	type:
	properties:
	message: string
	relevanceAlgorithm: RelevanceAlgorithm
	example: |
	{
	"message" : "Relevance algorithm updated",
	"relevanceAlgorithm" : {
	"id": "1",
	"algorithm" : "ALGORITHM REPRESENTATION"
	}
	}

delete:	description: Delete relevance algorithm having relevancealgorithmid={relevancealgorithmid} responses: 204: body: application/json: type:

	properties:
	message: string
	example: |
	{
	"message" : "Relevance algorithm deleted"
	}
	404:
	body:
	application/json:
	type:
	properties:
	message: string
	example: |
	{
	"message" : "Relevance algorithm not found"
	}