Apache UIMA Regular Expression Annotator Documentation

Authors: The Apache UIMA Development Community

Version 2.3.0

Incubation Notice and Disclaimer. Apache UIMA is an effort undergoing incubation at the Apache Software Foundation (ASF). Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.

Table of Contents

Introduction

The Regular Expression Annotator (RegexAnnotator) is an Apache UIMA analysis engine that detects entities such as email addresses, URLs, phone numbers, zip codes or any other entity that can be specified using a regular expression. For each entity that is detected an own annotation can be created or an already existing annotation can be updated with new features. To detect also more difficult and complex entities, the annotator provides some advanced filter capabilities and a rule definition syntax that can combine rules to a concept with a confidence value for each of the concept's rules.

Chapter 1. Processing Overview

To detect any kind of entity the RegexAnnotator must be configured using an external XML file. We call this file "concept file" since it contains the regular expressions and concepts that the annotator use during its processing to detect entities. In addition to the rules the concept file also contains the "entity result processing" that is done if an entity was detected. The "entity result processing" can either be the creation of new annotations or an update of an existing annotation with additional features. The types and features that are used to create new annotations have to be available in the UIMA type system.

After the concept file is created, the annotator XML descriptor have to be updated with the capabilities and maybe with the type system information from the concept file. The capability update is necessary that the UIMA framework can call the annotator also in complex annotator flows if the annotator is assembled with others to an analysis bundle. The UIMA type system update is only necessary if the used types are not available in the UIMA type system definition.

With the completion of the descriptor updates, the RegexAnnotator is ready to use. When starting the annotator, during the initialization the annotator reads the concept file and checks if all rules and concepts are valid and if all annotations types are defined in the UIMA type system. For each document that is processed the rules and concepts are executed in exactly the same order as defined in the concept file. The results and annotations created for a preceding rule are used by the following one since they are stored in the CAS.

Chapter 2. Concepts Configuration File

The RegexAnnotator can be configured using two levels of complexity.

The RuleSet definition is the easier way to define rules. Such a definition consists of a regular expression pattern and of annotations that should be created if the rule match an entity.

The Concept definition is the more complex way to define rules. Such a definition can consists of more than one regular expression rule that can be combined together and of a set of annotations that should be created if one of the rules has matched an entity.

The syntax for both definitions is the same, so you don't need to learn two configuration possibilities. The RuleSet definition is just available to have an easier and faster way to configure the annotator for simple tasks. If you have a RuleSet definition it is also possible to extend it with more and more features so that it becomes a real Concept definition.

2.1. RuleSet definition

The syntax of a simple RuleSet definition to detect email addresses is shown in the listing below:

```
<conceptSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="concept.xsd">

  <concept name="emailAddressDetection">
    <rules>
      <rule regEx="([a-zA-Z0-9!#$%*+'/=?^_-`{|}~.\x26]+)@
         ([a-zA-Z0-9._-]+[a-zA-Z]{2,4})"
        matchStrategy="matchAll" matchType="uima.tcas.DocumentAnnotation"/>
    </rules>
    <createAnnotations>
      <annotation id="emailAnnot" type="org.apache.uima.EmailAddress">
        <begin group="0"/>
        <end group="0"/>
      </annotation>
    </createAnnotations>
  </concept>

</conceptSet>
```

The definition above defines are simple concept with the name `emailAddressDetection`. The defined rule use `([a-zA-Z0-9!#$%*+'/=?^_-`{|}~.\x26]+)@([a-zA-Z0-9._-]+[a-zA-Z]{2,4})` as regular expression pattern that is matched on the covered text of the match type `uima.tcas.DocumentAnnotation`. As match strategy, `matchAll` is used that means that all matches for the pattern are used to create the annotations defined in the `<createAnnotations>` element. So for each match a

`org.apache.uima.EmailAddress` annotation is created that covers the match in the document text.

For additional annotation creation possibilities such as adding features to a created annotation, please refer to Section 2.5, “Annotation Creation” [10]

2.2. Concept definition

The syntax of a complex Concept definition to detect credit card numbers for the RegexAnnotator is shown in the listing below:

```
<conceptSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="concept.xsd">

    <concept name="creditCardNumberDetection" processAllRules="true">
      <rules>
        <rule ruleId="AmericanExpress"
              regEx="(((34|37)\d{2}[- ]?)(\d{6}[- ]?)\d{5})"
              matchStrategy="matchAll"
              matchType="uima.tcas.DocumentAnnotation"
              confidence="1.0"/>
        <rule ruleId="Visa"
              regEx="((4\d{3}[- ]?)(\d{4}[- ]?){2}\d{4})"
              matchStrategy="matchAll"
              matchType="uima.tcas.DocumentAnnotation"
              confidence="1.0"/>
        <rule ruleId="MasterCard"
              regEx="((5[1-5]\d{2}[- ]?)(\d{4}[- ]?){2}\d{4})"
              matchStrategy="matchAll"
              matchType="uima.tcas.DocumentAnnotation"
              confidence="1.0"/>
        <rule ruleId="unknownCardType"
              regEx="(([1-6]\d{3}[- ])(\d{4}[- ]){2}\d{4})|
                 ([1-6]\d{13,18})|([1-6]\d{3}[- ]\d{6}[- ]\d{5})"
              matchStrategy="matchAll"
              matchType="uima.tcas.DocumentAnnotation"
              confidence="1.0"/>
      </rules>
      <createAnnotations>
        <annotation id="creditCardNumber"
              type="org.apache.uima.CreditCardNumber"
              validate="org.apache.uima.annotator.regex.
                  extension.impl.CreditCardNumberValidator">
          <begin group="0"/>
          <end group="0"/>
          <setFeature name="confidence" type="Confidence"/>
          <setFeature name="cardType" type="RuleId"/>
        </annotation>
      </createAnnotations>
    </concept>
```

```
</conceptSet>
```

As you can see the Concept definition is a more complex RuleSet definition. The main differences are some additional features defined at the rule and the combination of rules within one concept. The new features for a rule are `ruleID` and `confidence`. If these features are specified, the feature values for these features can later be assigned to an annotation feature for a created annotation. In case we use the listing above as example this means that when the `org.apache.uima.CreditCardNumber` is created the value of the `confidence` feature of the rule that matched the document text is assigned to the annotation feature called `confidenceValue`. The same is done for the `ruleId` feature. With that you can later check your annotation confidence and you can see which rule was responsible for the annotation creation.

> **Note:** The annotation features for `Confidence` and `RuleId` have to be created manually in the UIMA type system. Given that it is possible to assign the `confidence` and `ruleId` feature values to any other annotation feature you have defined in the UIMA type system. Confidence features have to be of type `uima.cas.Float` and RuleId features have to be of type `uima.cas.String`.

The processing of a concept definition depends on the rule processing. The feature that controls the rule processing is called `processAllRules` and is specified at the `<concept>` element. By default this optional feature is set to `false`. This means that the concept processing starts with the first rule and goes on with the next one until a match was found. So in this processing mode, maybe only the first rule of a concept is evaluated if there a match was found. The other rules of this concept will be ignored in that case. This strategy should be used for example if your first concept rule has a strict pattern with a confidence of 1.0 and your second rule has a more lenient pattern with a confidence of 0.5. If the `processAllRules` feature is set to `true` all rules of a concept are processed independent of the matches for a previous rule.

2.3. Regex Variables

The regex variables allows to externalize parts of a regular expression to shorten them and make it easier to read. The externalized part of the expression is replaced with a regex variable. The variable syntax looks like `\v{weekdays}`, where `weekdays` is the variable name. The field for regex variables are mainly the separation of enumerations in a regular expression to make them easier to understand and maintain. But let's see how it works in the short example below.

A simple regular expression for a date like `Wednesday, November 28, 2007` can look like:

```
<concept name="Date" processAllRules="true">
 <rules>
  <rule regEx="(Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday),
   (January|February|March|April|May|June|July|August|September|October|
```

```
   November|December) (0[1-9]|[12][0-9]|3[01]), ((19|20)\d\d)"
   matchStrategy="matchAll" matchType="uima.tcas.DocumentAnnotation"/>
 </rules>
 <createAnnotations>
  <annotation type="org.apache.uima.Date">
   <begin group="0" />
   <end group="0" />
  </annotation>
 </createAnnotations>
</concept>
```

When using regex variables to externalize the weekdays and the months in this regular expression, it looks like:

```
<conceptSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://incubator.apache.org/uima/regex">

<variables>
 <variable name="weekdays"
   value="Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday"/>

 <variable name="months"
   value="January|February|March|April|May|June|July|August|September|
     October|November|December"/>
</variables>


<concept name="Date" processAllRules="true">
 <rules>
  <rule regEx="(\v{weekdays}), (\v{months}) (0[1-9]|[12][0-9]|3[01]),
     ((19|20)\d\d)"
     matchStrategy="matchAll" matchType="uima.tcas.DocumentAnnotation"/>
  </rules>
  <createAnnotations>
   <annotation type="org.apache.uima.Date">
    <begin group="0" />
    <end group="0" />
   </annotation>
 </createAnnotations>
</concept>

</conceptSet>
```

The regex variables must be defined at the beginning of the concept file next to the `<conceptSet>` element before the concepts are defined. The variables can be used in all concept definition within the same file.

The regex variable name can contain any of the following characters `[a-zA-Z_0-9]`. Other characters are not allowed.

2.4. Rule Definition

This paragraph shows in detail how to define a rule for a RuleSet or Concept definition and give you some advanced configuration possibilities for the rule processing.

The listing below shows an abstract rule definition with all possible sub elements and attributes. Please refer to the sub sections for details about the sub elements.

```
<rule ruleId="ID1" regEx="TestRegex" matchStrategy="matchAll"
    matchType="uima.tcas.DocumentAnnotation" featurePath="my/feature/path"
    confidence="1.0">

  <matchTypeFilter>
    <feature name="language">en</feature>
  </matchTypeFilter>

  <updateMatchTypeAnnotation>
    <setFeature name="language" type="String">$0</setFeature>
  </updateMatchTypeAnnotation>

  <ruleExceptions>
    <exception matchType="uima.tcas.DocumentAnnotation">
        ExceptionExpression
    </exception>
  </ruleExceptions>

</rule>
```

For each rule that should be added a `<rule>` element have to be created. The `<rule>` element definition has three mandatory features, these are:

- `regEx` - The regular expression pattern that is used for this rule. As pattern, everything supported by the Java regular expression syntax is allowed.
- `matchStrategy` - The match strategy that is used for this rule. Possible values are `matchAll` to get all matches, `matchFirst` to get the first match only and `matchComplete` to get matches where the whole input text match the regular expression pattern.
- `matchType` - The annotation type that is used to match the regular expression pattern. As input text for the match, the annotation span is used, but only if no additional `featurePath` feature is specified.

In addition to the mandatory features the `<rule>` element definition also has some optional features that can be used, these are:

- `ruleId` - Specifies the ID for this rule. The ID can later be used to add it as value to an annotation feature (see Section 2.5.3, “Annotation Features” [13]).
- `confidence` - Specifies the confidence value of this rule. If you have more than one rule that describes the same complex entity you can classify the rules with a

confidence value. This confidence value can later be used to add it as value to an annotation feature (see Section 2.5.3, “Annotation Features” [13]).

- `featurePath` - Specifies the feature path that should be used to match the regular expression pattern. If a feature path is specified, the feature path value is used to match against the regular expression instead of the match type annotation span. The defined feature path must be valid for the specified match type annotation type. The feature path elements are separated by "/".

  The listing below shows how to match a regular expression on the `normalizedText` feature of a `uima.TokenAnnotation`. So in this case, not the covered text of the `uima.TokenAnnotation` is used to match the regular expression but the `normalizedText` feature value of the annotation. The `normalizedText` feature must be defined in the UIMA type system as feature of type `uima.TokenAnnotation`.

```
<rule regEx="TestRegex" matchStrategy="matchAll"
    matchType="uima.TokenAnnotation" featurePath="normalizedText">
</rule>
```

2.4.1. Match Type Filter

```
<matchTypeFilter>
  <feature featurePath="language">en</feature>
</matchTypeFilter>
```

Match type filters can be used to filter the match type annotations that are used for matching the regular expression pattern. For example to use a rule only when the document language is English, as shown in the example above. Match type filters ever relate to the `matchType` that was specified for the rule.

The `<matchTypeFilter>` element can contain an arbitrary amount of `<feature>` elements that contains the filter information. But all specified `<feature>` elements have to be valid for the `matchType` annotation of the rule.

The feature path that should be used as filter is specified using the `featurePath` feature of the `<feature>` element. Feature path elements are separated by "/" e.g. my/feature/path. The specified feature path must be valid for the `matchType` annotation. The content of the `<feature>` element contains the regular expression pattern that is used as filter. To pass the filter, this pattern have to match the feature path value that is resolved using the match type annotation. In the example above the match type annotation has a UIMA feature called `language` that have to have the content `en`. If that is true, the annotation passed the filter condition.

2.4.2. Update Match Type Annotations With Additional Features

```
<updateMatchTypeAnnotation>
  <setFeature name="language" type="String">$0</setFeature>
</updateMatchTypeAnnotation>
```

With the `<updateMatchTypeAnnotation>` construct it is possible to update or set a UIMA feature value for the match type annotation in case a rule match was found. The `<updateMatchTypeAnnotation>` element can have an arbitrary amount of `<setFeature>` elements that contains the feature information that should be updated.

The `<setFeature>` element has two mandatory features, these are:

- `name` - Specifies the UIMA feature name that should be set. The feature have to be available at the `matchType` annotation of the rule.
- `type` - Specifies the UIMA feature type that is defined in the UIMA type system for this feature. Currently supported feature types are `String`, `Integer` and `Float`.

The optional features are:

- `normalization` - Specifies the normalization that should be performed before the feature value is assigned to the match type annotation. For a list of all built-in normalization functions please refer to Section 2.5.3.2, “Features Value Normalization” [14].
- `class` - Specifies the custom normalization class that should be used to normalize the feature value before it is assigned to the match type annotation. Custom normalization classes are used if the `normalization` feature has the value `Custom`. The normalization class have to implement the `org.apache.uima.annotator.regex.extension.Normalization` interface. For details about the feature normalization please refer to Section 2.5.3.2, “Features Value Normalization” [14].

The content of the `<setFeature>` element definition contains the feature value that should be set. This can either be a literal value or a regular expression capturing group as shown in the example above. A combination of capturing groups and literals is also possible.

2.4.3. Rule exception

```
<ruleExceptions>
  <exception matchType="uima.tcas.DocumentAnnotation">
      ExceptionPattern
  </exception>
</ruleExceptions>
```

With the <ruleExceptions> construct it is possible to configure exceptions to prevent matches for the rule. An exception is something similar to a filter, but on the higher level. For example take the scenario where you have several token annotations that are covered by a sentence annotation. You have written a rule that can detect car brands. The text you analyze has the sentence "Henry Ford was born 1863". When analyzing the text you will get a car brand annotation since "Ford" is a car brand. But is this the correct behavior? The work around that issue you can create an exception that looks like

```
<ruleExceptions>
  <exception matchType="uima.SentenceAnnotation">Henry</exception>
</ruleExceptions>
```

and add it to your car brand rule. After adding this, car brand annotations are only created if the sentence annotation that covers the token annotation does not contain the word "Henry".

The <ruleExceptions> element can have an arbitrary amount of <exception> elements to specify rule exceptions.

The <exception> element has one mandatory feature called matchType. The matchType feature specifies the annotation type the exception is based on. The concrete exception match type annotation that is used during the runtime is evaluated for each match type annotation that is used to match a rule. As exception annotation always the covering annotation of the current match type annotation is used. If no covering annotation instance of the exception match type was found the exception is not evaluated.

The content of the <exception> element specifies the regular expression that is used to evaluate the exception.

If the exception match is true, the current match type annotation is filtered out and is not used to create any matches and annotations.

2.5. Annotation Creation

This paragraph explains in detail how to create annotations if a rule has matched some input text. An annotation creation example with all possible settings is shown in the listing below.

```
<annotation id="testannot" type="org.apache.uima.TestAnnot"
 validate="CustomValidatorClass">
 <begin group="0" location="start"/>
 <end group="0" location="end"/>
 <setFeature name="testFeature1" type="String">$0</setFeature>
 <setFeature name="testFeature2" type="String"
  normalization="ToLowerCase">$0</setFeature>
 <setFeature name="testFeature3" type="Integer">$1</setFeature>
 <setFeature name="testFeature4" type="Float">$2</setFeature>
 <setFeature name="testFeature5" type="Reference">testannot1</setFeature>
```

```
 <setFeature name="confidenceValue" type="Confidence"/>
 <setFeature name="ruleId" type="RuleId"/>
 <setFeature name="normalizedText" type="String"
  normalization="Custom"
  class="org.apache.CustomNormalizer">$0</setFeature>
</annotation>
```

The <annotation> element has two mandatory features, these are:

- id - Specifies the annotation id for this annotation. If the annotation id is specified, it must be unique within the same concept. An annotation id is required if the annotation is referred by another annotation or if the annotation itself refers other annotations using a Reference feature.
- type - Specifies the UIMA annotation type that is used if an annotation is created. The used type have to be defined in the UIMA type system.

The optional features are:

- validate - Specifies the custom validator class that is used to validate matches before they are added as annotation to the CAS. For more details about the custom annotation validation, please refer to Section 2.5.2, “Annotation Validation” [12].

The mandatory sub elements of the <annotation> element are:

- <begin> - Specifies the begin position of the annotation that is created. For details about the <begin> element, please refer to Section 2.5.1, “Annotation Boundaries” [11].
- <end> - Specifies the end position of the annotation that is created. For details about the <end> element, please refer to Section 2.5.1, “Annotation Boundaries” [11].

The optional sub elements of the <annotation> element are:

- <setFeature> - set a UIMA feature for the created annotation. For details about the <setFeature> element, please refer to Section 2.5.3, “Annotation Features” [13]

2.5.1. Annotation Boundaries

When creating an annotation with the <annotation> element it is also necessary to define the annotations boundaries. The annotation boundaries are defined using the sub elements <begin> and <end>. The start position of the annotation is defined using the <begin> element, the end position using the <end> element. Both elements have the same features as shown below:

- group - identifies the capturing group number within the regular expression pattern for the current rule. The value is a positive number where 0 denotes the whole match, 1 the first capturing group, 2 the second one, and so on.

- `location` - indicates a position inside the capturing group, which can either be the position of the left parenthesis in case of a value `start`, or the right parenthesis in case of a value `end`. The `location` feature is optional. By default the `<begin>` element is set to `location="start"` and the `<end>` element to `location="end"`.

Note: When the rule definition defines a `featurePath` for a `matchType`, the annotation boundaries for the created annotation are automatically set to the annotation boundaries of the match input annotation. This must be done since the matching with a feature value of an annotation has no relation to the document text, so the only relation is the annotation where the feature is defined.

2.5.2. Annotation Validation

The custom annotation validation can be used to validate a regular expression match by using some java code before the match is added as annotation to the CAS. For example if your regular expression detects an ISBN number you can use the custom validation code to check if it is really an ISBN number by calculating the last check digit or if it is just a phone number.

To use the custom annotation validation you have to specify the validation class at the `validate` feature of the `<annotation>` element. The validation class must implement the `org.apache.uima.annotator.regex.extension.Validation` interface (Appendix B, *Validation Interface* [23]). The interface defines one method called `validate(String coveredText, String ruleID)`. The validate method is called by the annotator before the match is added as annotation to the CAS. Annotations are only added if the validate method returns `true`, otherwise the match is skipped. The `coveredText` parameter contains the text that matches the regular expression. The `ruleID` parameter contains the ruldId of the rule that creates the match. This can also be null if no ruleID was specified. The listing below shows a sample implementation of the validation interface.

```
package org.apache.uima.annotator.regex;

public class SampleValidator implements
 org.apache.uima.annotator.regex.extension.Validation {

   /* (non-Javadoc)
    * @see org.apache.uima.annotator.regex.extension.Validation
    *      #validate(java.lang.String, java.lang.String)
    */
   public boolean validate(String coveredText, String ruleID)
      throws Exception {

      //implement your custom validation, e.g. to validate ISBN numbers
      return validateISBNNumbers(coveredText);
   }
}
```

The configuration for this example looks like:

```
<annotation id="isbnNumber" type="org.apache.uima.ISBNNumber"
    validate="org.apache.uima.annotator.regex.SampleValidator">
 <begin group="0"/>
 <end group="0"/>
</annotation>
```

2.5.3. Annotation Features

With the `<setFeature>` element of `<annotation>` definition it is possible to set UIMA features for the created annotation. The mandatory features for the `<setFeature>` element are:

- `name` - Specifies the UIMA feature name that should be set. The feature name have to be a valid UIMA feature for this annotation and have to be defined in the UIMA type system.
- `type` - Specifies the type of the UIMA feature. For a list of all possible feature types please refer to Section 2.5.3.1, “Features Types” [14].

The optional features are:

- `normalization` - Specifies the normalization that should be performed before the feature value is assigned to the UIMA annotation. For a list of all built-in normalization functions please refer to Section 2.5.3.2, “Features Value Normalization” [14].
- `class` - Specifies the custom normalization class that should be used to normalize the feature value before it is assigned to the UIMA annotation. Custom normalization classes are used if the `normalization` feature has the value `Custom`. The normalization class have to implement the `org.apache.uima.annotator.regex.extension.Normalization` interface. For details about the feature normalization please refer to Section 2.5.3.2, “Features Value Normalization” [14].

The content of the `<setFeature>` element specifies the value of the UIMA feature that is set. As value a literal, a capturing group or a combination of both can be used. To add the value of a capturing group there are two ways to do it. The first notation is `$` followed by the capturing group number from 0 to 9 e.g. $0 for capturing group 0 or $7 for capturing group 7. The second notation to get the value of a capturing group are capturing group names. If the rule contains named capturing groups these groups can be accessed with `${matchGroupName}`. For the access of capturing groups greater than 9 capturing group names must be used. An example for capturing group names is shown below:

To add a name to a capturing group just add the following fragment `\m{groupname}` in front of the capturing group start parenthesis.

```
<concept name="capturingGroupNames">
   <rules>
      <rule ruleId="ID1"
```

```
         regEx="My \m{groupName}(named capturing group) example"
         matchStrategy="matchAll"
         matchType="uima.tcas.DocumentAnnotation"/>
   </rules>
   <createAnnotations>
      <annotation type="org.apache.uima.TestAnnot">
         <begin group="0"/>
         <end group="0"/>
         <setFeature name="testFeature0" type="String">
            ${groupName}
         </setFeature>
      </annotation>
   </createAnnotations>
</concept>
```

2.5.3.1. Features Types

When setting UIMA feature for an annotation using the `<setFeature>` element the feature type has to be specified according the the UIMA type system definition. The feature at the `<setFeature>` element to do that is called `type`. The list below shows all currently supported feature types:

- `String` - for `uima.cas.String` based UIMA features.
- `Integer` - for `uima.cas.Integer` based UIMA features.
- `Float` - for `uima.cas.Float` based UIMA features.
- `Reference` - to link a UIMA feature to another annotation. In this case the UIMA feature type have to be the same as the referred annotation type. To reference another annotation instance the `<setFeature>` content must have the annotation `id` as value of the referred annotation. The referred annotation instance is the created annotation of the current match.
- `Confidence` - to add the value of the `confidence` feature defined at the `<rule>` element to this feature. The UIMA feature have to be of type `uima.cas.Float`.
- `RuleId` - to add the value of the `ruleId` feature defined at the `<rule>` element to this feature. The UIMA feature have to be of type `uima.cas.String`.

Note: Float and Integer based feature values are converted using the Java NumberFormat for the current Java default locale. If the feature value cannot be converted the feature value is not set and a warning is written to the log. To prevent these warnings it may be useful to do a custom normalization of the numbers before they are added to the feature.

2.5.3.2. Features Value Normalization

Before assigning a feature value to an annotation it is possible to do a normalization on the feature value. This normalization can be useful for example to normalize a detected

email addresses to lower case before it is added to the annotation. To normalize a feature value the `normalization` feature of the `<setFeature>` element is used. The built-in normalization functions are listed below. Additionally the RegexAnnotator provides an extension point that can be implemented to add a custom normalization.

The possible build-in functions that are specified as feature value of the `normalization` feature are listed below:

- `ToLowerCase` - normalize the feature value to lower case before it is assigned to the annotation.
- `ToUpperCase` - normalize the feature value to upper case before it is assigned to the annotation.
- `Trim` - remove all leading and trailing whitespace characters from the feature value before it is assigned to the annotation.

Built-in normalization configuration:

```
<setFeature name="normalizedFeature" type="String"
 normalization="ToLowerCase">$0</setFeature>
```

In case of a custom normalization, the `normalization` feature must have the value `Custom`, and an additional feature of the `<setFeature>` element called `class` have to be specified containing the full qualified class name of the custom normalization implementation. The custom normalization implementation have to implement the interface `org.apache.uima.annotator.regex.extension.Normalization` (Appendix C, *Normalization Interface* [25]) which defines the `normalize` method to normalize the feature values. A sample implementation with the corresponding configuration is shown below.

Custom normalization implementation:

```
package org.apache.uima;

public class CustomNormalizer
  implements org.apache.uima.annotator.regex.extension.Normalization {

   /* (non-Javadoc)
    * @see org.apache.uima.annotator.regex.extension.Normalization
    *  #normalize(java.lang.String, java.lang.String)
    */
   public String normalize(String input, String ruleId) {

      //implement your custom normalization
      String result = ...
      return result;
   }
```

Custom normalization configuration:

```
<setFeature name="normalizedFeature" type="String"
 normalization="Custom" class="org.apache.uima.CustomNormalizer">
  $0
</setFeature>
```

Chapter 3. Annotator Descriptor

The RegexAnnotator analysis engine descriptor contains some processing information for the annotator. The processing information is specified as configuration parameters. This chapter we explain in detail the possible descriptor settings.

3.1. Configuration Parameters

The RegexAnnotator has the following configuration parameters:

- `ConceptFiles` - This parameter is modeled as array of Strings and contains the concept files the annotator should use. The concept files must be specified using a relative path that is available in the UIMA datapath or in the classpath. When you use the UIMA datapath, you can use wildcard expressions such as `rules/*.rule`. These kinds of wildcard expressions will not work when rule files are discovered via the classpath.

```
<nameValuePair>
  <name>ConceptFiles</name>
  <value>
    <array>
      <string>subdir/myConcepts.xml</string>
      <string>SampleConcept.xml</string>
    </array>
  </value>
</nameValuePair>
```

3.2. Capabilities

In the capabilities section of the RegexAnnotator descriptor the input and output capabilities and the supported languages have to be defined.

The input capabilities defined in the descriptor have to comply with the match types used in the concept rule file that is used. For example the `uima.SentenceAnnotation` used in the rule below have to be added to the input capability section in the RegexAnnotator descriptor.

```
<rules>
  <rule regEx="SampleRegex" matchStrategy="matchAll"
      matchType="uima.SentenceAnnotation"/>
</rules>
```

In the output section, all of the annotation types and features created by the RegexAnnotator have to be specified. These have to match the output types and features declared in the `<annotation>` elements of the concept file. For example the `org.apache.uima.TestAnnot` annotation and the

`org.apache.uima.TestAnnot:testFeature` feature used below have to be added to the output capability section in the RegexAnnotator descriptor.

```
<createAnnotations>
  <annotation type="org.apache.uima.TestAnnot">
    <begin group="0"/>
    <end group="0"/>
    <setFeature name="testFeature" type="String">$0</setFeature>
  </annotation>
</createAnnotations>
```

If there are any language dependent rules in the concept file the languages abbreviations have to be specified in the `<languagesSupported>`element. If there are no language dependent rules available you can specify `x-unspecified` as language. That means that the annotator can work on all languages.

For the short examples used above the capabilities section in the RegexAnnotator descriptor looks like:

```
<capabilities>
  <capability>
    <inputs>
      <type>uima.SentenceAnnotation</type>
    </inputs>
    <outputs>
      <type>org.apache.uima.TestAnnot</type>
      <feature>org.apache.uima.TestAnnot:testFeature</feature>
    </outputs>
    <languagesSupported>
      <language>x-unspecified</language>
    </languagesSupported>
  </capability>
</capabilities>
```

Appendix A. Concept File Schema

The concept file schema that is used to define the concept file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://incubator.apache.org/uima/regex"
   xmlns="http://incubator.apache.org/uima/regex"
   elementFormDefault="qualified">
 <!--
  * Licensed to the Apache Software Foundation (ASF) under one
  * or more contributor license agreements.  See the NOTICE file
  * distributed with this work for additional information
  * regarding copyright ownership.  The ASF licenses this file
  * to you under the Apache License, Version 2.0 (the
  * "License"); you may not use this file except in compliance
  * with the License.  You may obtain a copy of the License at
  *
  *   http://www.apache.org/licenses/LICENSE-2.0
  *
  * Unless required by applicable law or agreed to in writing,
  * software distributed under the License is distributed on an
  * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
  * KIND, either express or implied.  See the License for the
  * specific language governing permissions and limitations
  * under the License.
 -->

  <xs:element name="conceptSet">
 <xs:complexType>
   <xs:sequence>
  <xs:element ref="concept" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
  </xs:element>

  <xs:element name="concept">
 <xs:complexType>
   <xs:sequence>
  <xs:element ref="rules" minOccurs="1" maxOccurs="1"/>
  <xs:element ref="createAnnotations" minOccurs="1" maxOccurs="1"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:string" use="optional"/>
 </xs:complexType>
  </xs:element>

  <xs:element name="createAnnotations">
 <xs:complexType>
   <xs:sequence>
  <xs:element ref="annotation" minOccurs="1" maxOccurs="unbounded"/>
   </xs:sequence>
 </xs:complexType>
```

```
 </xs:element>

 <xs:element name="rules">
<xs:complexType>
  <xs:sequence>
 <xs:element ref="rule" minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
 </xs:element>

 <xs:element name="rule">
<xs:complexType>
  <xs:all>
 <xs:element ref="matchTypeFilter" minOccurs="0" maxOccurs="1"/>
 <xs:element ref="updateMatchTypeAnnotation" minOccurs="0" maxOccurs="1"/>
 <xs:element ref="ruleExceptions" minOccurs="0" maxOccurs="1"/>
  </xs:all>
  <xs:attribute name="regEx" type="xs:string" use="required"/>
  <xs:attribute name="matchStrategy" use="required">
    <xs:simpleType>
   <xs:restriction base="xs:string">
     <xs:enumeration value="matchFirst"/>
  <xs:enumeration value="matchAll"/>
  <xs:enumeration value="matchComplete"/>
   </xs:restriction>
 </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="matchType" type="xs:string" use="required"/>
  <xs:attribute name="featurePath" type="xs:string" use="optional" />
  <xs:attribute name="ruleId" type="xs:string" use="optional"/>
  <xs:attribute name="confidence" type="xs:decimal" use="optional"/>
</xs:complexType>
 </xs:element>

 <xs:element name="matchTypeFilter">
<xs:complexType>
  <xs:sequence>
 <xs:element ref="feature" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
 </xs:element>

 <xs:element name="ruleExceptions">
<xs:complexType>
  <xs:sequence>
    <xs:element ref="exception" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
 </xs:element>

 <xs:element name="exception">
<xs:complexType>
  <xs:simpleContent>
```

```
 <xs:extension base="xs:string">
   <xs:attribute name="matchType" type="xs:string" use="required"/>
 </xs:extension>
  </xs:simpleContent>
</xs:complexType>
 </xs:element>

 <xs:element name="feature">
<xs:complexType>
  <xs:simpleContent>
 <xs:extension base="xs:string">
   <xs:attribute name="featurePath" type="xs:string" use="required"/>
 </xs:extension>
  </xs:simpleContent>
</xs:complexType>
 </xs:element>

 <xs:element name="annotation">
<xs:complexType>
  <xs:sequence>
 <xs:element ref="begin" minOccurs="1" maxOccurs="1"/>
 <xs:element ref="end" minOccurs="1" maxOccurs="1"/>
 <xs:element ref="setFeature" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="type" type="xs:string" use="required"/>
  <xs:attribute name="validate" type="xs:string" use="optional" />
</xs:complexType>
 </xs:element>

 <xs:element name="updateMatchTypeAnnotation">
<xs:complexType>
  <xs:sequence>
    <xs:element ref="setFeature" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
 </xs:element>

 <xs:element name="begin">
<xs:complexType>
  <xs:attribute name="group" use="required" type="xs:integer"/>
  <xs:attribute name="location" use="optional" default="start">
    <xs:simpleType>
      <xs:restriction base="xs:string">
     <xs:enumeration value="start"/>
     <xs:enumeration value="end"/>
   </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
 </xs:element>

 <xs:element name="end">
```

```
 <xs:complexType>
   <xs:attribute name="group" use="required" type="xs:integer"/>
   <xs:attribute name="location" use="optional" default="end">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="start"/>
   <xs:enumeration value="end"/>
    </xs:restriction>
 </xs:simpleType>
  </xs:attribute>
</xs:complexType>
 </xs:element>

 <xs:element name="setFeature">
<xs:complexType>
  <xs:simpleContent>
 <xs:extension base="xs:string">
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="type" use="required">
     <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="String"/>
   <xs:enumeration value="Integer"/>
   <xs:enumeration value="Float"/>
   <xs:enumeration value="Reference"/>
   <xs:enumeration value="Confidence"/>
   <xs:enumeration value="RuleId"/>
    </xs:restriction>
  </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="normalization" use="optional">
     <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Custom" />
   <xs:enumeration value="ToLowerCase" />
   <xs:enumeration value="ToUpperCase" />
   <xs:enumeration value="Trim" />
    </xs:restriction>
  </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="class" type="xs:string" use="optional" />
 </xs:extension>
  </xs:simpleContent>
</xs:complexType>
 </xs:element>
</xs:schema>
```

Appendix B. Validation Interface

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
package org.apache.uima.annotator.regex.extension;


/**
 * The Validation interface is provided to implement a custom validator
 * that can be used to validate regular expression matches before
 * they are added as annotations.
 */
public interface Validation {

/**
 * The validate method validates the covered text of an annotator and
 * returns true or false whether the annotation is correct or not.
 * The validate method is called between a rule match and the
 * annotation creation. The annotation is only created if the method
 * returns true.
 *
 * @param coveredText covered text of the annotation that should be
 *        validated
 * @param ruleID ruleID of the rule which created the match
 *
 * @return true if the annotation is valid or false if the annotation
 *         is invalid
 *
 * @throws Exception throws an exception if an validation error occurred
 */
public boolean validate(String coveredText, String ruleID)
   throws Exception;

}
```

Appendix C. Normalization Interface

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one
 * or more contributor license agreements.  See the NOTICE file
 * distributed with this work for additional information
 * regarding copyright ownership.  The ASF licenses this file
 * to you under the Apache License, Version 2.0 (the
 * "License"); you may not use this file except in compliance
 * with the License.  You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed under the License is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.  See the License for the
 * specific language governing permissions and limitations
 * under the License.
 */
package org.apache.uima.annotator.regex.extension;


/**
 * The Normalization interface was add to implement a custom normalization
 * for feature values before they are assigned to an anntoation.
 */
public interface Normalization {

/**
 * Custom feature value normalization. This interface must be implemented
 * to perform a custom normalization on the given input string.
 *
 * @param input input string which should be normalized
 *
 * @param ruleID rule ID of the matching rule
 *
 * @return String - normalized input string
 */
public String normalize(String input, String ruleID) throws Exception;
}
```