3 - Defining workflows in YAML

This tutorial introduces how Kurator-Akka workflows are specified in YAML files.

What is YAML?

YAML is a plain text format for representing data organized as lists of values and sets of key-value pairs (mappings). The values in these lists and in the key-value pairs can themselves be lists or mappings. YAML is a superset of JSON (every JSON document is a valid YAML document) that uses white space (rather than braces and quotes) to organize data, and is thus easy to read.

Kurator-Akka uses YAML to represent workflow definitions so that workflows can be specified and executed without using any software development tools (other than a text editor), and so that other programs can generate workflow specifications simply by producing a YAML file. Kurator-Akka uses yaml-spring-loader (written by Scott McPhillips for the RestFlow system) to parse YAML files and to create (using Spring) the workflow components the files describe.

Note that grouping and nesting of data in YAML is achieved by aligning and indenting the text in the file. Spaces are used for alignment and indenting. Tabs are never used in a YAML file.

See http://yaml.org/ for more information about YAML and a list of libraries in C/C++, Ruby, Python, Java, Perl, C#, and other languages for composing and parsing YAML.

Example workflow

We will use the `hello.yaml` workflow from the Kurator-Akka distribution to illustrate how workflows are specified in YAML. You can extract this YAML file from the kurator-akka jar file using the unzip command (the -j option prevents any directories from being created during inflation):

```
$ unzip -j kurator-akka-0.2-executable.jar org/kurator/akka/samples/hello.yaml
Archive:  kurator-akka-0.2-executable.jar
  inflating: hello.yaml
$
```

The contents of `hello.yaml` are as follows:

```
imports:

  - classpath:/org/kurator/akka/actors.yaml

components:

  - id: GreetingSource
    type: ConstantSourceActor
    properties:
      parameters:
        value: Hello World!

  - id: GreetingPrinter
    type: PrinterActor
    properties:
      listensTo:
        - !ref GreetingSource

  - id: HelloWorldWorkflow
    type: Workflow
    properties:
      actors:
        - !ref GreetingSource
        - !ref GreetingPrinter
      parameters:
        greeting:
          actor: !ref GreetingSource
          parameter: value
```

Structure of a YAML workflow definition file

Inspect the contents of `hello.yaml` above. Kurator-Akka expects each YAML workflow definition file to have a mapping (set of key-value pairs) as the top-level data structure. A colon in YAML indicates that the preceding string is a key, and that the following value or block of text is the value assigned to that key. The valid keys of this top-level mapping are *imports*, *types*, and *components*, with the result that Kurator-Akka workflow definition files have up to three top-level sections: an imports section, a types section, and a components section. The `hello.yaml` file does not have a *types* section.

The imports section (the block of text following the imports line) is for providing a list of other YAML files to be included in the current workflow definition. List items are preceded by a dash. The components section provides a list of the workflow components comprising the workflow. We will focus on the workflow components for the remainder of this tutorial page.

Workflow components

The components section of hello.yaml contains declarations for two actors and for the workflow as whole. Actors are the active, data processing components of workflow. The workflow itself is considered a component as well, with the workflow component declaration containing references to the actor components in it.

Each component in `hello.yaml` is described by a mapping with three keys: *id*, *type*, and *properties*. The *id* of a component is an arbitrary text string that is used to uniquely identify it. The *id* also enables components to refer to each other.

The *type* of a component refers to a declaration either in the type section of the current YAML file or in a YAML file included (directly or indirectly) in the *imports* section. Type declarations associate each component type with a Java class, and will be covered in a later tutorial.

Finally, each component has a set of *properties* that represents its configuration in the current workflow.

Examing the components in hello.yaml

The first component in `hello.yaml` is the actor with id `GreetingSource`:

```
 - id: GreetingSource
   type: ConstantSourceActor
   properties:
     parameters:
       value: Hello World!
```

This declaration states that the component named GreetingSource is an instance of the ConstantSourceActor. ConstantSourceActor emits a value (or sequence of values) at the beginning of a workflow run. The value emitted by a ConstantSourceActor is specified by the *value* parameter. Parameters represent a subset of component properties that are available to the component at run-time. Properties not specified as parameters are used by the Kurator-Akka framework when building the workflow and are not visible to running components. Here, GreetingSource is configured to emit the string, 'Hello World!' when the workflow starts running.

The second component in hello.yaml is the actor with id GreetingPrinter:

```
- id: GreetingPrinter
   type: PrinterActor
   properties:
     listensTo:
       - !ref GreetingSource
```

`GreetingPrinter` is an instance of `PrinterActor.` A `PrinterActor` writes any data it receives to an output stream (which defaults to stdout of the process running the workflow). The `listensTo` property of `GreetingPrinter` is assigned a list containing a single element with the value `!ref GreetingSource`. Actors in a workflow generally receive the data they process from other actors in the same workflow, and the `listensTo` property on an actor lists the actors that this actor receives data from. The `!ref` keyword indicates that the string following it is not a literal value but a reference to (identifier of) a component defined elsewhere in the workflow. Here, `GreetingActor` is configured to receive any data emitted by `GreetingSource`, and thus will receive the string 'Hello World!' emitted by `GreetingSource` when the workflow is run.

The final component of in `hello.yaml` is identified as `HelloWorldWorkflow`:

```
 - id: HelloWorldWorkflow
   type: Workflow
   properties:
     actors:
       - !ref GreetingSource
       - !ref GreetingPrinter
     parameters:
       greeting:
         actor: !ref GreetingSource
         parameter: value
```

`HelloWorldWorkflow` is an instance of `Workflow.` A `Workflow` component is used to specify which actors defined in the yaml file (or in imported yaml files) comprise a single workflow. A `Workflow` also gives values for properties that affect the workflow as a whole. The *actors* property is assigned a list of references to the actors in the workflow, while the *parameters* property is used to expose parameters on individual actors in the workflow. Parameters exposed in this way are configurable by users of the workflow.

Here, `HelloWorldWorkflow` is configured to comprise two actors, `GreetingSource` and `GreetingPrinter`. The *value* parameter of the `GreetingSource` actor is exposed as the *greeting* parameter of the workflow. See the final section of Tutorial 2 - Running workflows for an example run of `hello.yaml` with an assignment to the *greeting* parameter of the workflow.