Ingest Workflow Language

(See also Jon Corson-Rikert's examples.)

The ingest workflow language was added in August 2008 as a simple way of scripting actions that would otherwise require manual interaction with the Ingest Tools page. At the time it was imagined that sequences of ingest tool actions might be saved as a workflow and edited through the GUI, but this functionality was never implemented.

Workflows are specified in RDF. As there is no GUI support for creating or editing the workflows, it is typically most attractive to write the workflows in N3 syntax.

Executing an existing workflow

To run a workflow, the RDF statements describing the workflow must be visible to the web application. This can be done by selecting "attach to webapp" from the "Manage Jena Models" page for a model containing a workflow definition, or by loading the workflow directly into the main model using "Add /Remove RDF Data" from the Site Administration page.

Selecting "Execute Workflow" from the main Ingest Tools menu shows a list of visible workflows. A workflow may then be executed from the beginning, or from any of its component steps (e.g., to avoid redundant re-processing of data in early workflow steps).

Creating a new workflow

RDF workflow descriptions use the following namespaces:

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix w:       <http://vitro.mannlib.cornell.edu/ns/vitro/rdfIngestWorkflow#> .
@prefix s:       <http://vitro.mannlib.cornell.edu/ns/vitro/0.7/sparql#> .
@prefix vitro:   <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#> .
@prefix ex:      <http://example.org/myWorkflow#> .
```

If you are using N3 syntax, you can simply copy the above prefix declarations into an empty text file to begin your workflow. Note that the last line defines the namespace for the resources that make up the workflow. You should change this namespace to one using a domain name under your control.

The workflow resource

Each workflow begins with a resource representing that workflow itself:

```
ex:MyFirstWorkflow
   a            w:Workflow ;
   rdfs:label   "My First Workflow" ;
   w:firstStep  ex:step1 .
```

This defines a resource of type w:Workflow (a class recognized by the workflow engine) that has a label and which points to the first step in the workflow. Note that the URIs you use to identify the pieces of the workflow are entirely arbitrary: "ex:MyFirstWorkflow" and "ex:step1" could just as easily have been "ex:sadf1289723" or "ex:n2222" or "ex:ClearTheModelsFirst".

Workflow steps

A workflow is just a linked list of workflow steps. Each workflow step has a label and a property pointing to the action that is to performed at this step. This arrangement means that a particular action can be defined once and performed at multiple different steps in a workflow.

```
ex:MyFirstWorkflow
   a            w:Workflow ;
   rdfs:label   "My First Workflow" ;
   w:firstStep  ex:step1 .

ex:step1
   a            w:WorkflowStep ;
   rdfs:label   "Do something first" ;
   w:action     ex:someAction ;
   w:nextStep   ex:step2 .

ex:step2
   a            w:WorkflowStep ;
   rdfs:label   "Now do something else" ;
   w:action     ex:anotherAction ;
   w:nextStep   ex:step3 .

ex:step3
   a            w:WorkflowStep ;
   rdfs:label   "Do a third thing" ;
   w:action     ex:aThirdAction .
```

Note how the w:action property links a workflow step to the action that is to be performed at that step, and w:nextStep links to the next step in the workflow. Execution of a workflow ends when there is a workflow step that does not have any w:nextStep property.

Actions

There are eight different types of actions, which use different properties to specify their parameters:

```
w:ClearModelAction
w:AddModelsAction
w:SubtractModelsAction
w:ExecuteSparqlConstructAction
w:SmushResourcesAction
w:NameBlankNodesAction
w:SplitPropertyValuesAction
w:ProcessPropertyValueStringsAction
```

Actions work on one or more RDF models, which must be visible to the ingest tools either in the default database or by using the "Connect DB" page. Models are represented as resources of type w:Model:

```
   ex:myWorkingModel
       a           w:Model ;
       rdfs:label  "working model" ;
       w:modelName "working model" .
```

The value of the w:modelName property should match the model name that appears on the Manage Jena Models page. The label is arbitrary.

Using blank nodes to simplify syntax

When actions are not reused across multiple workflow steps, it can be convenient to describe them using blank nodes. This avoids having to assign a separate URI to each action, and also allows them to be written inline:

```
crw:CreateFullNetIDs
     a w:WorkflowStep ;
     rdfs:label "Create Cornell NetID property"@en-US ;
     w:action [
       a w:SPARQLCONSTRUCTAction ;
       w:sourceModel crw:cheResponseModel ;
       w:destinationModel crw:cheResponseModel ;
           w:sparqlQuery
            [ s:queryStr """ PREFIX che: <http://vitro.mannlib.cornell.edu/ns/ingest/CHE#>
                        PREFIX vivo: <http://vivo.library.cornell.edu/ns/0.1#>
                        PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
           CONSTRUCT {
               ?s vivo:CornellemailnetId ?o
           } WHERE {
               ?s che:cheresponse_Netid ?o
           }""" ]
     ] .
```

Notice how the w:action property does not link to the URI of an action but to a blank node that describes the action. (In addition, another blank node is used inside the action description to avoid assigning a URI to a particular SPARQL query.)

ClearModelAction

This action removes all the statements from a model.

parameters: w:sourceModel

example:

```
crw:ClearWorkingModel
     a             w:ClearModelAction ;
     rdfs:label    "clear working model" ;
     w:sourceModel ex:workingModel .
```

Note that somewhere in the workflow file ex:workingModel must be described:

```
ex:workingModel
       a           w:Model ;
       rdfs:label  "working model" ;
       w:modelName "working model" .
```

AddModelsAction

This action adds the statements in one model (the "model to add") to another model (the "destination model"). Note that there currently seems to be a bug in this action where a "source model" must be supplied, but nothing is actually done with the source model.

parameters: w:destinationModel, w:modelToAdd, w:sourceModel (see note above)

SubtractModelsAction

This action subtracts the statements in one model (the "model to subtract") from another model (the "source model") and saves the difference in a third model (the "destination model").

parameters: w:sourceModel, w:modelToSubtract, w:destinationModel

ExecuteSparqlConstructAction

This action runs a SPARQL CONSTRUCT query on a model (the "source model") and adds the constructed triples to another model (the "destination model").

parameters: w:sourceModel, w:destinationModel, w:sparqlQuery (see example for specification of query)

Multiple source models may be specified for this action, by adding additional statements using w:sourceModel .

example:

```
crw:CreateFullNetIDs
     a w:WorkflowStep ;
     rdfs:label "Create Cornell NetID property"@en-US ;
     w:action [
       a w:SPARQLCONSTRUCTAction ;
       w:sourceModel crw:cheResponseModel ;
               w:sourceModel crw:anotherModel ;
       w:destinationModel crw:cheResponseModel ;
           w:sparqlQuery
            [ s:queryStr """ PREFIX che: <http://vitro.mannlib.cornell.edu/ns/ingest/CHE#>
                        PREFIX vivo: <http://vivo.library.cornell.edu/ns/0.1#>
                        PREFIX fn: <http://www.w3.org/2005/xpath-functions#>
           CONSTRUCT {
               ?s vivo:CornellemailnetId ?o
           } WHERE {
               ?s che:cheresponse_Netid ?o
           }""" ]
     ] .
```

SmushResourcesAction

This action rewrites URIs so that all resources with the same value for a given property will share the same URI.

parameters: w:sourceModel, w:smushOnProperty, w:destinationModel

Multiple values of w:sourceModel may be supplied for this action.

The value of the w:smushOnProperty statement is a literal containing the URI of the property on which to smush, e.g.

```
ex:smushAction
   a                  w:SmushResourcesAction ;
   w:sourceModel      ex:model1 ;
   w:destinationModel ex:model2 ;
   w:smushOnProperty  "http://www.example.org/ontology/employeeID" .
```

NameBlankNodesAction

This action turns blank nodes into randomly-generated URIs

parameters: w:sourceModel, w:destinationModel, w:uriPrefix

Multiple values of w:sourceModel may be supplied for this action.

The value of the w:uriPrefix statement is a literal containing the namespace for the URIs **plus** the initial non-numeric portion of the local name.

For example,

```
   w:uriPrefix "http://example.org/individual/n"
```

will cause the action to rename blank nodes with URIs that look like

```
   <http://example.org/individual/n23568>
   <http://example.org/individual/n41>
   <http://example.org/individual/n9156662>
```

SplitPropertyValuesAction

This action splits statements where object is a literal containing delimited values into separate statements, one for each value.

parameters: w:sourceModel, w:destinationModel, w:splitRegex, w:originalProperty, w:newProperty

The value of the w:originalProperty is a literal containing the URI of the property with the delimited values and the value of the w:newProperty is a literal containing the URI of the property to be used for each split value.

example:

```
   ex:SplitValues
       a                 w:SplitPropertyValuesAction ;
       w:sourceModel     ex:model1 ;
       w:destinationModel ex:model2 ;
       w:originalProperty "http://example.org/ontology/departmentIDs" ;
       w:newProperty     "http://example.org/ontology/departmentID" ;
       w:splitRegex      "," .
```

This will transform the following RDF in the source model

```
    ex:something
           <http://example.org/ontology/departmentIDs> "CHEM, BIO, BIO-PL, FOODS" .
```

into the following RDF in the destination model

```
    ex:something
           <http://example.org/ontology/departmentID> "CHEM" ;
           <http://example.org/ontology/departmentID> "BIO" ;
           <http://example.org/ontology/departmentID> "BIO-PL" ;
           <http://example.org/ontology/departmentID> "FOODS" .
```

ProcessPropertyValuesAction

This action will run an arbitrary method on a Java class available on the application classpath to transform string values of a given property. The method should take a single String as a parameter and return a String.

parameters: w:sourceModel, w:destinationModel, w:originalProperty, w:newProperty, w:processorClass, w:processorMethod

example:

```
crw:AppendCornellEdu
   a w:WorkflowStep ;
   rdfs:label "Append @Cornell.edu to net ids"@en-US ;
   w:action [
       a w:ProcessPropertyValueStringsAction ;
       w:sourceModel crw:cheResponseModel ;
       w:destinationModel crw:cheResponseModel ;
       w:originalProperty [
           a w:Literal ;
           w:literalValue "http://vivo.library.cornell.edu/ns/0.1#CornellemailnetId"
           ] ;
       w:newProperty [
           a w:Literal ;
           w:literalValue "http://vivo.library.cornell.edu/ns/0.1#CornellemailnetId"
           ] ;
       w:processorClass [
           a w:Literal ;
           w:literalValue           "edu.cornell.mannlib.vitro.bjl23.ingest.hr.HRCornellEmailProcessor"
           ] ;
       w:processorMethod [
           a w:Literal ;
           w:literalValue "process"
           ]
   ] .
```