Getting Started with Kepler Provenance 2.5

November 2015

Getting Started with the Kepler Provenance Module

The Provenance module was created to capture and query workflow execution history.

Table of Contents

1. Introduction

This guide introduces the main components and functionality of the Provenance module. The Provenance module is an add-on module suite for Kepler, a software application for the analysis and modeling of scientific data, and it provides functionality for capturing and querying workflow execution history stored locally on your computer.

2. Downloading and Installing the Provenance Module

From the Kepler application menu select Tools => Module Manager. From the Module Manager dialog, select the Available Modules tab. Select 'provenance-2.5' from the list of Available Suites, then click the right arrow button to move the provenance-2.5 suite to the list of Selected Modules. Click the 'Apply and Restart' button to retrieve the provenance suite and restart Kepler.

3. Capturing Provenance

Once you have installed the Provenance module, you will see a button on the toolbar with a "P" as shown in Figure 1. By default, provenance capturing is on, and may be turned off by pushing this button. The button will turn red, denoting recording is now off. If you have the Reporting suite installed, both the Workflow Run Manager and Report Designer are disabled unless provenance capturing is turned on. Further, provenance can only be captured for workflows that have SDF, DDF, or PN directors.

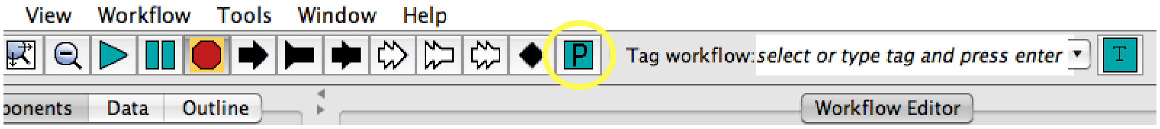


Figure 1: Provenance Button

3.1. Configuring Storage Location

By default, provenance information is written to an HSQL database located in `$HOME/KeplerData/modules/provenance/provenanceDB`. You can change the location of this database or store provenance in a MySQL, Oracle, or PostgreSQL database by editing the Provenance module's configuration file, which is located in `$HOME/KeplerData/modules/provenance/configuration/configuration.xml`. (If this file does not exist, you can edit the provenance configuration file in the Kepler installation directory). Table 1 describes several relevant fields in this file.

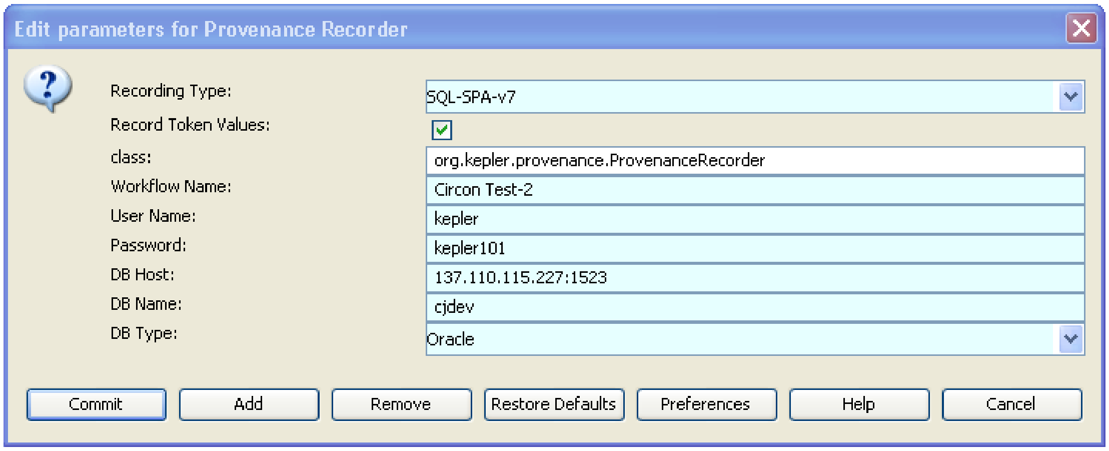
Table 1: Provenance Configuration Fields

Field	Description
DB Host	The hostname of the database server.
DB Port	The port number for the database server.
DB User Name	The user name for the database.
Password	The password for the database.
DB Type	The type of database: HSQL, MySQL, Oracle, PostgreSQL.
DB Name	The name of the database. If the database type is HSQL, this is the filename in $HOME/KeplerData/modules/provenance. (Absolute paths can also be used). If the database type is MySQL, this is the schema name. If the database type is Oracle, this is the SID name.
DB Table Prefix	This string will be prepended to all tables in the provenance schema. If you want to add the provenance tables into an existing database, this field can be used to prevent name collisions.

When the provenance system makes the first connection to the database, it checks if the provenance tables already exist. If they are not found, they are automatically created.

The configuration file provides the default location to store provenance information for all workflow runs. Alternatively, you can specify these settings on a per-workflow basis by adding the Provenance Recorder to the workflow canvas:

1. Drag and drop the Provenance Recorder from the actor library to the workflow canvas. (You can search for "provenance" in the Search Components field).
2. Double-click on the Provenance Recorder on the canvas and a dialog will appear (Figure 2) allowing you to edit the configuration parameters. When you are done, push the "Commit" button to save your changes.
3. Finally, click on the "P" button on the toolbar to turn off the default provenance settings.



Edit parameters for Provenance Recorder

Recording Type: SQL-SPA-v7
Record Token Values: ☑
class: org.kepler.provenance.ProvenanceRecorder
Workflow Name: Circon Test-2
User Name: kepler
Password: kepler101
DB Host: 137.110.115.227:1523
DB Name: cjdev
DB Type: Oracle

Figure 2: Provenance Recorder Configuration Dialog

4. The Provenance Manager Command-Line Program

The Provenance Manager is a command-line program that provides access to the information stored in the provenance database. The Provenance Manager is *$HOME/KeplerData/kepler.modules/provenance-2.5.0/prov-manager.sh* on Mac and Linux, and *%USERPROFILE%/KeplerData/kepler.modules/provenance-2.5.0/prov-manager.bat* on Windows.

To list all the workflow executions in the database:

```
prov-manager.sh -l
```

The Provenance Manager can export workflow executions from the database. The provenance is serialized as PROV JSON[1] and written to a KAR file.

```
prov-manager -o output.kar [-all | -run n] [-delete]
-all      export all workflows runs.
-run n    export run n.
-delete   delete workflow runs in database after export.
```

Workflow executions can also be imported:

```
prov-manager -i input.kar [-force]
-force  attempt to import runs despite any missing
dependencies.
```

The Provenance Manager has other command-line options that are not described in this document. Run *prov-manager* with –*h* to see them all.

5. Provenance Query API

Provenance information is stored in a relational schema, and the Provenance module includes a Java API to access this information. The API is described in the interface `org.kepler.provenance.Queryable` and implemented in the class `org.kepler.provenance.sql.SQLQueryV8`. Below are several example queries and their implementation; the object *query* is an instance of `Queryable`.

1. What workflow names does the database contain?

```
List<String> names = query.getWorkflows();
```

2. What are the run ids for workflow "a"?

```
List<Integer> ids = query.getExecutionsForWorkflow("a");
```

3. What are the ids for all data transferred between actors for run 2?

```
List<Integer> ids =
               query.getTokensForExecution(2, true);
```

4. What is the workflow definition (MoML) for run 2?

```
String moml = query.getMoMLForExecution(2);
```

5. What are the names and values of each parameter for run 2?

```
Map<String,String> map =
```

[1] PROV JSON is described here.

```
                query.getParameterNameValuesForExecution(2);
```

Section 6.4 shows how to answer these queries using SQL.

6. Relational Schema

The relational schema represents three types of information: the contents or specification of workflows, how these specifications change over time, and events that occur during workflow execution. This section describes the tables in the relational schema for each of these areas.

6.1. Specification Tables

The workflow specification records information about actors, directors, and parameters, and ports in each workflow.

6.1.1. Actor

Name	Type	References	Recorded By
class	varchar		regActor()
id	long	entity(id)	regActor()

This table maps an actor to its implementation class.

6.1.2. Director

Name	Type	References	Recorded By
class	varchar		regDirector()
id	long	entity(id)	regDirector()

This table maps a director to its implementation class.

6.1.3. Entity

Name	Type	References	Recorded By
deleted	boolean		regNNN()
display	varchar		regNNN()
id	long		regNNN()
name	varchar		regNNN()
prev_id	long		regNNN()
type	varchar		regNNN()
wf_change_id	long	workflow_change(id)	regNNN()
wf_id	long	workflow(id)	regNNN()

Each entity in the workflow, such as actors, ports, parameters, relations, etc., is represented by a row in this table. *Name* is the fully qualified name of the entity in the workflow relative to the workflow name. The *display* column contains the display name of the entity if it is different from its name; otherwise *display* is empty. The *type* column denotes the type of entity such as “actor”, or “port”.

When a parameter’s value changes, a new row is added to this table and the *parameter* table: the *entity(id)* of the new row can be used to find the new parameter value in *parameter(value)*. Previous values of the parameter can be found by following the *entity(prev_id)* ids. Currently, *deleted* is always false.

6.1.4. Parameter

Name	Type	References	Recorded By
id	long	entity(id)	regParameter()
type	varchar		regParameter()
value	varchar		regParameter()

This table stores parameter types and values, including all previous values for each parameter.

6.1.5. Port

Name	Type	References	Recorded By
direction	int		regPort()
id	long	entity(id)	regPort()
multiport	boolean		regPort()

Actors read and write tokens via ports. The *direction* column holds an enumeration value that denotes if the port is an input, output, or input-output port. *Multiport* is true if the port allows more than one connection to it.

6.1.6. Tag

Name	Type	References	Recorded By
id	long		
searchstring	varchar		
type	varchar		
urn	varchar		
wf_exec_lsid	varchar	workflow_exec(lsid)	

This table stores tags associated with either workflows or workflow executions. The *urn* contains the URI of the ontology concept being tagged, which is normally appended with another "#" and the label of the concept; if the concept cannot be resolved, it can still be displayed (if it turns up in search results, for instance) by its normal human-readable representation. The *type* designates either a workflow or workflow execution. In the latter case, *searchstring* is searched against when searching through executions or workflows, and *wf_exec_lsid* refers to the execution's LSID.

6.1.7. Workflow

Name	Type	References	Recorded By
id	long		provenance events
lsid	varchar		provenance events
name	varchar		provenance events

This table contains a row for each workflow in the provenance database. Each workflow has a unique LSID and id. A workflow may optionally have a (non-unique) name. The LSID in table does **not** include the revision.

6.2. Workflow Evolution

6.2.1. Workflow_change

Name	Type	References	Recorded By
id	long		evolution/specificationStart()
host_id	varchar		evolution/specificationStart()
time	datetime		evolution/specificationStart()
user	varchar		evolution/specificationStart()
wf_id	long	workflow(id)	evolution/specificationStart()

Each row in this table corresponds to a user-driven workflow update. Currently, this is only recorded when specifying the workflow structure the first time, or when parameter values change. In the future, this table could be used to record when actors, directors, etc. are added or removed from the workflow.

6.3. Workflow Execution

6.3.1. Actor_fire

Name	Type	References	Recorded By
actor_id	long	actor(id)	actorFire()
end_time	datetime		actorFire()
id	long		actorFire()
start_time	datetime		actorFire()
wf_exec_id	long	workflow_exec(id)	actorFire()

This table records information about actor firings for a particular actor (*actor_id*) in a particular workflow execution (*wf_exec_id*).

6.3.2. Associated_data

Name	Type	References	Recorded By
data_id	varchar	data(md5)	addFileForExecution()
id	long		addFileForExecution()
name	varchar		addFileForExecution()
val	varchar		addFileForExecution()
wf_exec_id	long	workflow_exec(id)	addFileForExecution()

This table provides a mechanism for files and other data objects to be stored for a given workflow execution. It is primarily used to store files that capture the state of the workflow at the time of execution and/or the results of the execution (in the case of the reporting module). It is similar to how the workflow MoML is saved at each execution - providing a snapshot of that point in time.

When interacting (inserting and querying records) from this table one or more metadata *name/value* pairs can be utilized. Note that in the simplest case there will be a single row for a single data object with one *name/value* pair. In future uses of the table there may be multiple rows for the same data object that differ only by the metadata *name/value* pairs. Imagine a case in which the same file is stored for different reasons by different modules. Similarly, multiple metadata rows will be useful for limiting the results of querying for particular data objects for a given execution in cases where there are many associated data files for that execution.

6.3.3. Data

Name	Type	References	Recorded By
contents	BLOB		portEvent(), executionStart()
md5	varchar		portEvent(), executionStart()
truncated	boolean		portEvent(), executionStart()

This table contains data used by the workflow including tokens, and workflow MoMLs. If the data was too large to be stored in the *contents* column, *truncated* is true*.*

6.3.4. Error

Name	Type	References	Recorded By
entity_id	long		
id	long		
exec_id	long	workflow_exec(id)	
message	varchar		

This table stores errors that occur during workflow executions. The error string is in *message* and *entity_id* is the workflow component that created the error. Both *message* and *entity_id* are optional.

6.3.5. Parameter_exec

Name	Type	References	Recorded By
parameter_id	long	parameter(id)	executionStart()
wf_exec_id	long	workflow_exec(id)	executionStart()

This table records the values of all parameters at the **start** of a workflow execution. (Parameter values may change during the execution; use the *parameter* table to access additional values).

6.3.6. Port_event

Name	Type	References	Recorded By
channel	long		portEvent()
data	varchar		portEvent()
data_id	varchar	data(md5)	portEvent()
file_id	varchar	data(md5)	portEvent()
fire_id	long	actor_fire(id)	portEvent()
id	long		portEvent()
port_id	long	port(id)	portEvent()
time	datetime		portEvent()
type	varchar		portEvent()
write_event_id	long		portEvent()

Each token read or write is stored as a row in this table. A port event occurred at *time*, on port *port_id,* and on *channel* from actor firing *fire_id*. If the size of the token's string value is less than or equal to 4096 characters, then the value is stored in *data.* Otherwise, the value is stored in *data(contents)* and referenced by *data_id.* The token's class name is in *type*. If the token is a string containing a file name, and the size of the file is less than maxFileInclusionSizeKB (a parameter in the provenance configuration file), then the contents of the file are stored in the *data* table and a reference to the contents is stored in *file_id*. If the port event represents a read, *write_event_id* is the *port_event*(*id)* of the port event that generated the token; otherwise (the port event is a write) *write_event_id* is -1. For read events, *data*, *data_id*, and *type* are not set since they are already provided in the write event.

6.3.7. Workflow_exec

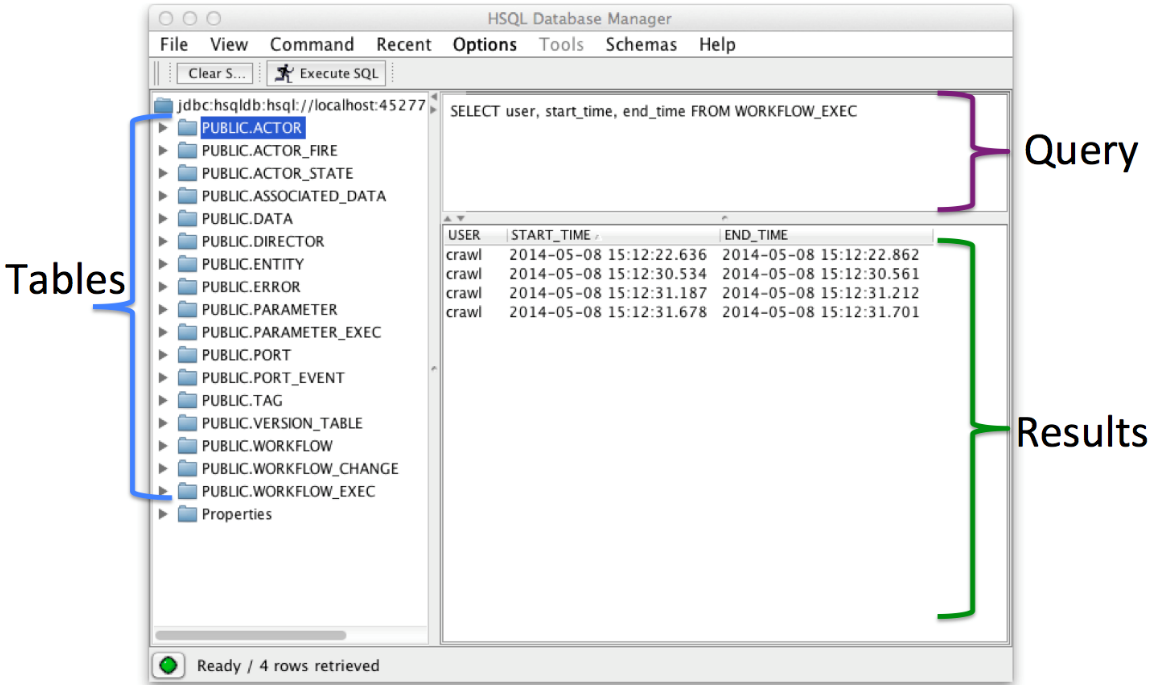
Name	Type	References	Recorded By
annotation	varchar		executionStart()
derived_from	varchar		insertRunReferralList()

end_time	datetime		executionStop()
host_id	varchar		executionStart()
lsid	varchar		executionStart()
id	long		executionStart()
module_dependencies	varchar		executionStart()
start_time	datetime		executionStart()
user	varchar		executionStart()
wf_contents_id	varchar	data(md5)	executionStart()
wf_full_lsid	varchar		executionStart()
wf_id	long	workflow(id)	executionStart()

Each row in this table corresponds to a workflow execution; it describes which workflow executed, who ran it, the start and stop times, and a unique LSID assigned to the execution. The *wf_contents_id* column references the workflow MoML (as it exists at the start of the execution). Additionally, an annotation string may be specified for each execution. The workflow's LSID (including revision) is stored in *wf_full_lsid*. The *module_dependencies* column contains the list of currently running Kepler modules.

6.4. The HSQL Query Browser

Kepler writes provenance information to an HSQL database by default. You can start a graphical query browser to view the contents of this database; e.g., you can execute the SQL queries described in the next section. To start the HSQL query browser, run *$HOME/KeplerData/kepler.modules/provenance-2.5.0/prov-hsql.sh* on Mac or Linux, or *%USERPROFILE%/KeplerData/kepler.modules/provenance-2.5.0/prov-hsql.bat* on Windows.



```
SELECT user, start_time, end_time FROM WORKFLOW_EXEC
```

```
USER   START_TIME               END_TIME
crawl  2014-05-08 15:12:22.636  2014-05-08 15:12:22.862
crawl  2014-05-08 15:12:30.534  2014-05-08 15:12:30.561
crawl  2014-05-08 15:12:31.187  2014-05-08 15:12:31.212
crawl  2014-05-08 15:12:31.678  2014-05-08 15:12:31.701
```

Figure 3: The HSQL Query Browser

The query browser is shown in Figure 3. The pane on the left side shows all the tables in the database. In the top right, you can enter SQL commands, and the results are shown in the bottom right.

6.5. Example SQL Queries

1. What workflow names does the database contain?

```
SELECT name
FROM workflow
```

2. What are the run ids for workflow “a”?

```
SELECT wf_exec.id
FROM workflow_exec wf_exec, workflow wf
WHERE wf_exec.wf_id = wf.id AND wf.name = ‘a’
```

3. What are the ids for all data transferred between actors for run 2?

```
SELECT pe.data_id
FROM port_event pe, actor_fire af
WHERE pe.fire_id = af.id AND af.wf_exec_id = 2
```

4. What is the workflow definition (MoML) for a run 2?

```
SELECT d.contents
```

```
FROM data d, workflow_exec wf_exec
WHERE d.md5 = wf_exec.wf_contents_id AND wf_exec.id = 2
```

5. What are the names and values of each parameter for run 2?

```
SELECT e.name, p.value
FROM entity e, parameter p, parameter_exec pe
WHERE pe.parameter_id = e.id AND pe.parameter_id = p.id AND
pe.wf_exec_id = 2
```

For additional SQL queries, see `org.kepler.provenance.sql.SQLQueryV8`.