Karma Provenance Service
User Manual V3.2

April 14, 2011

This document contains instructions for building and installing the Karma service, v3.2, which provides core capability to store provenance events to a database and returns results. Karma Provenance Tool is licensed under Apache License, Version 2.0 (the "License") (http://www.apache.org/licenses/LICENSE-2.0). The code is copyrighted and copyright owned by The Trustees of Indiana University. Karma is a product of the Data to Insight Center at Indiana University. See http://pti.iu.edu/d2i/provenance for more information.

Table of Contents

==========

1. Introduction

==========

Karma provenance tool is modeled on a client server paradigm where a Karma service runs persistently and accepts and stores incoming provenance info and responds to queries. Karma clients either serve users or sit at the applications to be instrumented, and serve to capture provenance events as they occur. These events are sent to the Karma server (called a service) through either a event service bus (RabbitMQ) or a web service interface. The Karma server can be built to be a standalone daemon when running with RabbitMQ. If configured for web service, it can be built into an .aar file and dropped in an Axis2 container. The Axis2 container can either be standalone or deployed into an Apache Tomcat server.

Karma v3.2 is an upgrade from v3.1.1. Main new features include:

1) Supporting annotation related notifications. Karma now supports addition of new annotations to existing entities/files (called objects). This feature is added so that additional annotations from other sources can be appended at some later point in time thereby making an enriched provenance store. New annotations can be added to an object in two ways:
i) By URI: matches the URI of the object and adds annotations to it. This can be used to add annotations to objects which have similar URIs.
ii) By ID: if the internal id of the object (to uniquely identify the object in the repository) is known, it adds annotations to the object. It can be used to add annotations to a specific object only.

Apart from this, support for adding annotations to multiple objects has also been added. This enables adding annotations to multiple objects using a single notification.

2) Linux shell scripts to run Karma Messaging Client test cases. Whenever a user implements his/her own test code using Karma Messaging Client API, he/she can run the code using KarmaRun.sh script without importing the libraries and setting up environment variables.

Bug fix in v3.1.1: The previous version's getOPM API call for retrieving OPM xml has now been overhauled and renamed getWorkflowGraph().

There's a minor database schema change from v3.1.1 to v3.2. The annotation_value field in all the annotation tables have TEXT datatype to allow larger values. If v3.1.1 database schema is used, then follow the instructions on step-4 and 5 to upgrade it to v3.2. However, if v3.1 or earlier version of database schema is used, an additional upgrade is needed. To upgrade an existing database v3.1 to the latest schema, please follow these steps:

1. Backup the existing database

```
mysqldump -u <karmauser> -p <karma_db_name> --add-drop-table --complete-insert --disable-keys > karma_3.1_dump.sql
```

2. Check your old Karma database version. If v3.1.1 database schema is used, follow the instructions on step 4 and 5 to upgrade. Otherwise, locate the upgrade sql script db_schema_upgrade_from_3.1.0_to_3.1.1.sql, which can be found in the config/ directory.

3. apply this script to the v3.1 karma database

```
cd config
mysql -u <karmauser> -p <karma_db_name> < db_schema_upgrade_from_3.1.0_to_3.1.1.sql
```

4. Locate the alter-db script, karma_alter_db_annotation_value.sql, which is present in the config/ directory:

5. apply this script to the v3.1.1 karma database

```
cd config
mysql -u <karmauser> -p <karma_db_name> < karma_alter_db_annotation_value.sql
```

To upgrade the server deployment to v3.2, simply follow the installation instructions and replace the existing deployment.

2. Software Dependencies

==========

Karma v3.1.1 has been tested with the following software packages on which it has a dependency. These packages will need to be installed separately:

2.1. Basic

1) Apache ANT v1.6 or higher (for building Karma from source)
http://ant.apache.org

2) mySQL Database Community Server v5.1
http://dev.mysql.com

3) mySQL Connector/JDBC v5.1 or higher
http://dev.mysql.com/downloads/connector/j/

4) Java Development Kit (JDK) v5 or v6
http://java.sun.com

JSR 173 API (required if using JDK5)

JSR 173 API is integrated into JDK6. However, if the build environment uses JDK5, the JSR 173 API jar must be manually included in the classpath. This jar comes with XML Beans v2.3.0 package

5) Apache XML Beans v2.3.0
http://xmlbeans.apache.org
Look for v2.3.0 from any binary distribution mirror sites

2.2 Web Service vs. Messaging System

Choose either group 6 or group 7 sets of instructions. To host Karma as a web service, 6.1 and 6.2 are required. To host Karma as a standalone server using RabbitMQ Messaging System, 7.1 and 7.2 are required.

6) Web Service

6.1) Apache Tomcat Server v5.5x or v6 (If to run Karma as an Apache Axis2 web service)
http://tomcat.apache.org

6.2) Apache Axis2/Java v1.4 or v1.5x WAR (Web Archive) Distribution (If to run Karma as an Apache Axis2 web service)
http://ws.apache.org/axis2

7) Standalone Server with Integrated Messaging System

7.1) RabbitMQ Server.
http://www.rabbitmq.com

7.2) Erlang is required for building RabbitMQ. Install a recent version of Erlang/OTP from
http://www.erlang.org

Please refer to the corresponding documentation of the third party software packages for installation instructions. See also sections IV and V respectively.

Karma v3.2 also conforms to Open Provenance Model (OPM) v1.1.
http://eprints.ecs.soton.ac.uk/18332/1/opm.pdf

3. Setup Karma Database

==========

After you have successfully installed and launched a mySQL database instance, please follow these steps to setup the Karma database.

1. Log into mySQL as an administrator:

```
mysql -u root -p
```

2. Create a database, preferably named "karma"

```
CREATE DATABASE karma;
```

3. Create login credentials for karma and grant permissions

```
GRANT ALL ON karma.* TO 'karmauser'@'localhost' IDENTIFIED BY 'karmapwd'
```

In the example above, 'karmauser' and 'karmapwd' are just examples. DO NOT use these verbatim; choose something more secure instead, especially for the password.

This example also assumes that the database is hosted on the same node as Karma. If this is not the case, please specify the host onto which Karma is hosted:

```
GRANT ALL ON karma.* TO 'karmauser'@'dedicated-node.iu.edu' IDENTIFIED BY 'karmapwd';
```

4. Use the provided database schema definition to define tables. The schema definition file can be found in the config/ directory of the Karma service distribution, and is named karma_db_schema.sql

```
cd karma-3.2
mysql -u root -p karma < config/karma_db_schema_v3.2.sql
```

4. Configure Karma Properties

==========

The Karma server depends on a number of properties for the correct operation as well as performance tweaking. The distribution package contains a sample properties file, which can be found in config/karma.properties. Please use this sample file to configure Karma according to the deployment environment. Below is the detailed explanation of each of these properties.

4.1 Required properties

–- These properties must be properly set regardless of how Karma is hosted (as an Axis2 web service or standalone server using RabbitMQ).

log4j.properties.path - path to a log4j.properties file. Karma uses log4j to log information. A sample log4j.properties file is also provided with the distribution package, in the config/ directory. This log4j.properties file defines two separate loggers, karmlog and karmconsole. karmlog is a rolling file

appender that writes log messages to a series of files on disk, whereas karmconsole is a console appender that writes log messages to screen. Please modify log4j.properties to change settings such as the name and location of the log file etc.

database.location – URI of karma database

database.username – username for logging into karma database

database.password – password for logging into karma database

conn.pool.init.size – initial size of karma database connection pool. Creating database connection is expensive. Karma uses a connection pool to deliver faster performance.

conn.pool.max.size – maximum size of karma database connection pool.

conn.time.to.live.ms – number of milliseconds a database connection in the pool lives before being replaced by a new connection. This is to ensure the database connections in the pool do not timeout. Please choose a moderate number. If the number is too small, each connection gets replaced too frequently and causes performance overhead; on the other hand, if the number is too large, the connection may timeout and also cause performance penalty.

async.processor.thread.count - number of asynchronous raw (unprocessed) notification processors to run. Karma asynchronously processes ingested notifications to avoid server overload.

raw.notif.process.batch.size – maximum number of raw (unprocessed) notifications to fetch from the database for each access.

raw.notif.cache.size - number of raw notifications each asynchronous processor takes after they are fetched from the database.

The following formula may serve as a guideline for specifying the above 3 properties:
raw.notif.cache.size = raw.notif.batch.size / async.processor.thread.count

annotation.definition.scope.count - number of known annotation definition scope settings. This property must be followed by *N* pairs of **annotation.definition.property.***x* and **annotation.definition.scope.***x*

annotation.definition.property.*x* - name or URI of annotation property *x*, where 1<=*x*<=*N*, *N* being the value given to **annotation.definition.scope.count**

annotation.definition.scope.*x* - scope of annotation property *x*, where 1<=*x*<=*N*. The value of this property **must be one of the following tokens**:

OPM_ANNOTATION - this annotation is defined by OPM
KARMA_ANNOTATION - this annotation is defined by Karma

EXTERNAL_SOURCE - this annotation is defined by some other external source

4.2 Optional properties

--Applicable only if Karma uses event service bus (RabbitMQ) communication

The following 5 properties are used to connect to a RabbitMQ Server. If the instance of RabbitMQ already exists, contact the system admin for details.

messaging.username -- RabbitMQ Username
messaging.password -- RabbitMq Password
messaging.hostname -- Hostname that hosts RabbitMQ Server
messaging.hostport -- Port number on the RabbitMQ Server
messaging.virtualhost -- The virtual host is created for administrative purposes. Each connection (and all channels inside) must be associated with a single virtual host. Each virtual host comprises of its own namespace, a set of exchanges, message queues and all associated objects. The default value is "/".

The following 3 properties are used to configure how to send the Notifications to Karma Server.

messaging.exchangename -- A message routing agent. It can be durable (our system uses durable), temporary, and auto-deleted. Each message is delivered to each qualifying queue. The default value here is "KarmaExchange".

messaging.queuename -- Named "Weak FIFO" buffer. The default value is "KarmaQueue".
messaging.routingkey -- In our implementation, we use direct exchange type. Same routingkey is used on both publisher and subscriber sides. The default value here is "KarmaKey"

The following 2 properties are used for the Karma Server to reconnect to the RabbitMQ Server in case of failures.

messaging.retry.interval -- Karma server will wait for N seconds (retry interval) before each retry if connection between itself and RabbitMQ Server is lost. The default value for N is 5.
messaging.retry.threshold -- Karma Server will retry up to X times (retry threshold) to create the connection between itself and RabbitMQ Server. The default value for X is 5.

5. Host Karma as Axis2 Web Service

==========

Currently there are two options to host Karma. One is to host Karma as an Apache Axis2 Web Service. The other option is to host Karma as a stand-alone server using rabbitMQ for messaging. This section is on how to host Karma as a web service.

5.1 Setup Web Service Container with Axis2

Assuming that you have successfully installed and launched an instance of Apache Tomcat server and successfully downloaded a copy of Apache Axis2 WAR distribution, follow these steps to deploy Axis2:

1. Unzip Axis2 WAR distribution package

```
unzip axis2-1.5.1-war.zip
```

2. Copy the WAR file into the webapps/ directory under tomcat

```
cp axis2-1.5.1-war/axis2.war ~/apache-tomcat-5.5/webapps/
```

Apache Tomcat server will automatically expand the WAR file and create a directory named axis2/

5.2 Building Karma Service as Axis2 Service

Take the following steps to build Karma as a Web Service.

1. Generate Java code and XmlBeans from the WSDL and XML Schema

```
ant wsdl2java
```

2. Edit the following file to make necessary configuration changes

generated/resources/services.xml

2.1) The service should be configured to run with "application" scope. This is done by adding the clause

scope="application"

to the line that reads

<service name="KarmaService">

so it reads

<service name="KarmaService" scope="application">

2.2) Add the following custom initialization parameters required by Karma.
The values shown below are just examples. Please edit the actual values

to suit your deployment environment. Explanation of each parameter follows.

```
<parameter name="karma.properties.file.path">
      /home/karma/karma.properties
</parameter>
```

2.3) Explanation of the parameters

karma.properties.file.path - path to the properties file configured in Section 4.

3. Save the modified services.xml file and continue build

```
ant karma.webservice
```

This generates a karma.aar file under build/lib/ directory

Note: if you would like to start over, you may use the following command. BE AWARE that this command will delete all built files, including the services.xml you have modified

```
ant clean
```

5.3 Deploying Karma Service

Now that Karma has been built as an axis2 archive (AAR), it needs to be deployed.

1. Copy the aar file into axis2's service directory

```
cp karma.aar ~/apache-tomcat-5.5/webapps/axis2/WEB-INF/services/
```

2. karma.aar does not contain the mySQL JDBC jar. This is intentional to avoid duplicate or even conflicting versions of JDBC jar files being deployed into a Tomcat/axis2 server that already has JDBC jar deployed.

However, if your Tomcat server does not yet have mySQL JDBC jar file, please copy the one you downloaded earlier into axis2's lib directory:

```
cp mysql-connection-java-5.1.x-bin.jar ~/apache-tomcat-5.5/webapps/axis2/WEB-INF/lib
```

6. Host Karma as Standalone Server using RabbitMQ

==========

This section is on how to host Karma as a standalone server using RabbitMQ for messaging.

6.1 Install RabbitMQ and Erlang/OTP

If hosting RabbitMQ on Windows, complete bundle and installation documents can be retrieved at http://www.rabbitmq.com/install.html

If hosting RabbitMQ on platforms other than Windows, download Erlang/OTP from http://www.erlang.org/download.html and follow the installation guide here. Then download RabbitMQ server from http://www.rabbitmq.com/download.html and follow the installation documents inside the package.

6.2 Build and Deploy Karma Service as Standalone Server

Follow the ant command below to build Karma Service as Standalone Server using RabbitMQ Messaging System.

```
cd ${Karma-Service-Core}
build-standalone.sh
```

6.3 Run Karma Service as Standalone Server

```
cd ${Karma-Service-Core}/bin
./KarmaServer.sh ${Karma-Service-Core}/config/karma.properties
```

Default log file (karma.log) can be found at execution directory.

7. Karma Messaging Client

==========

This section is on how to use Karma Service Client when the Karma Server is hosted as standalone server and using RabbitMQ for messaging. This section assumes you are setting up new Karma provenance server. If you will be connecting to an existing Karma server that already has a RabbitMQ messaging bus running, please contact your RabbitMQ messaging bus administrator and configure the karma.properties file.

7.1 Configure and Build Karma Service Client

```
cd ${Karma-Client-Core}
vi build.properties
```

Point the karma.client.properties to a valid properties path. Default is ${Karma-Client-

Core}/config/karma.properties

```
ant
```

Client API can be found in ${Karma-Client-Core}/build/lib/karma-client.jar

7.2 Use Karma Service Client

Edit ${Karma-Client-Core}/config/karma.properties properly. See Section 4.2.

We offer 2 options to use Karma Service Client.

7.2.1 Command Line:
We provide two sets of commands to ingest, query and test against Karma Server: Linux shell scripts, and ant command.

a) Notification Ingest:

```
./bin/sendNotification.sh config/karma.properties ${path_to_notification_xml_file}
```

or,

```
ant sendNotification -Dnotification=${path_to_notification_xml_file}
```

The client will not return ingest successful/fail status when ingesting via Karma Messaging Client.
b) Query:

```
./bin/query.sh config/karma.properties ${path_to_query_xml_file}
```

or,

```
ant query -Dquery=${path_to_query_xml_file}
```

c) To run any main method in the karma client package

```
./bin/KarmaRun.sh <Main Class> <Arguments>
```

or,

```
ant run -DrunClass=< Main Class > -DrunArgs="<Arguments>"
```

7.2.2 Write your own application and invoke Karma Service Client API inside karma-client.jar
API specification can be found under ${Karma-Client-Core}/doc

7.3 Karma Service Client Use Cases

7.3.1 Test Cases through command line:

There're 3 sample notification and 1 query files under sample/ directory.

*Send these notifications using the following command.

```
./bin/sendNotification.sh config/karma.properties ${Karma-Client-Core}/samples/notification/sendingResponse.xml
```

```
./bin/sendNotification.sh config/karma.properties ${Karma-Client-Core}/samples/notification/
sendingResponseStatus.xml

./bin/sendNotification.sh config/karma.properties ${Karma-Client-Core}/samples/notification/ serviceInvoked.xml
```

or,

```
ant sendNotification -Dnotification=${Karma-Client-Core}/samples/notification/sendingResponse.xml

ant sendNotification -Dnotification=${Karma-Client-Core}/samples/notification/sendingResponseStatus.xml

ant sendNotification -Dnotification=${Karma-Client-Core}/samples/notification/serviceInvoked.xml
```

*Query

```
./bin/query.sh config/karma.properties ${Karma-Client-Core}/samples/notification/getWorkflowGraphRequest.xml
```

or,

```
ant query -Dquery=${Karma-Client-Core}/samples/notification/getWorkflowGraphRequest.xml
```

7.3.2 Test Cases through Karma Service Client API calls

There're several test codes for ingest and query the Karma Server, under the test/ directory. These sample codes can be compiled and run through command line.

```
./bin/KarmaRun.sh <Main Class> <Arguments>
```

or,

```
ant run -DrunClass=< Main Class > -DrunArgs="<Arguments>"
```

8. Karma Web Service Client

==========

This section is on how to use Karma Service Client when Karma Server is hosted as an Apache Axis2 Web Service.

8.1 Create Karma Axis2 Web Service Client Stub

The following command may be used to generate the Karma Axis2 web service client stub and types xmlbeans jar:

```
ant jar
```

Generated jars can be found in the following directory:
dist/lib/client-stub-types.jar

8.2 Compile Karma Axis2 Web Service Client Test Code

This test client consists of code that can be used to test the ingest API and Query API of the Karma Axis2 Web Service. The test code consists of ingest calls and query calls that has been preconfigured to make use of dummy data values. These values should be modified accordingly.

The following command is used to compile the Karma Axis2 web service test client code:

```
ant test.compile
```

Compiled classes can be found in the following directory:
dist/test/classes/

Alternatively, the compiled test client code can be packaged into a jar file using the following command:

```
ant test.jar
```

The resulting jar file can be found in the following directory:
dist/lib/client-testcode.jar

8.3 Executing Karma Axis2 Web Service Client Test Code

The following command script can be used to execute the Karma Axis2 web service test client.

```
./run.sh test-client
```

Two parameters should be modified according to the execution environment before executing the script. These parameters are as follows:

KARMA_WEBSERVICE_URL:	This parameter should point to the Karma Axis2 web service Service URL.
THREAD_COUNT:	This parameter is used to specify the number of threads that the test client should use.