LIGADATA

OnLEP 1.0 Setup and Running Guide

Overview

This guide will help you get the LigaDATA OnLEP engine (Online Learning Engine Platform) and Metadata.API installed and ready to use.

The OnLEP engine processes incoming streaming data, transforms it into messages or containers, and processes it according to the ruleset you have supplied in your models. It then produces decision or alert data that can be acted upon programmatically.

The Metadata.API defines the CRUD operations (create, read, update, delete) for metadata objects supported by this system. The metadata objects include Types, Functions, Concepts, Derived Concepts, MessageDefinitions, and Model Definitions. All functions take String values as input in XML or JSON Format and return a JSON string of ApiResult object.

Prerequisites:

- Approximately 300 MB free
    - Access to: https://github.com/ligaDATA/RTD_ReadOnly OR the RTD install package

Windows users:

- 7-zip installed (http://www.7-zip.org/), choose for 32bit or 64bit.

  Note: for OpenSource, we should use zip or tar.gz)
- Scala installed, v2.10.4 and install sbt: http://www.scala-sbt.org/download.html
- Install the Java Development Kit (JDK 1.7.1 or greater), which should install JRE – if not:
- Java installed (Java runtime version 1.7 or later, which can be downloaded here.)
- Download and install Kafka: http://kafka.apache.org/
- Download and install Zookeeper: http://zookeeper.apache.org/releases.html#download
- The definition of the data models and rule sets used in your business that need to be created as models.

Assumptions:

At least one instance of each of the following is running:

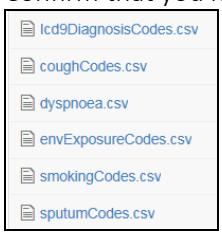
- Zookeeper Service
- Kafka
- Cassandra OR HBase

NOTE: Cassandra and HBase are optional - you will need some input/output to the engine.

Linux users: It is suggested that Linux users run the services/scripts under screens (http://www.howtoforge.com/linux_screen), so that if the connection is lost, they will still be running. This will likely become a service in a later release.)

To Test OnLEP on a single node, install the following:

1. From GitHub, clone: https://github.com/ligaDATA/RTD_ReadOnly
   a. Navigate to the folder:
   <localCloneDirectory>/RTD_ReadOnly/trunk/SampleApplication/Medical/SampleData
   b. Confirm that you have the following files:



Linux & Mac users:

2. Run engine install script:

   <localCloneDirectory>/RTD_ReadOnly/trunk/SampleApplication/Medical/bin/installOnLEP_Medical.sh
   </path/to/install/location> <localCloneDirectory>/RTD_ReadOnly/trunk

   Example: <localCloneDirectory>/ligadata/trunk/SampleApplication/Medical/bin/installOnLEP_Medical.sh /tmp/OnLEPInstall <$HOME>/ligadata/trunk

3. Modify the following configuration files to match your local environment – The first 2 configuration files are used in running your application, and the last configuration file is used during the MetadataAPI execution (all these files can be found at the following location:

   <localCloneDirectory>/RTD_ReadOnly/trunk/SampleApplication/Medical/Configs)

   Please note that it is best practice to copy these files to a separate directory <localConfigDirectory> to make sure that the changes are not overwritten when an application is re-installed or new version is downloaded from the github.

   - COPD.properties – See example in Appendix. Keep as is for single node test. This file is used in the OnLEP engine.
   - MetadataAPIConfig.properties – This file is used when running the Metadata.API application.

   For example, you'll need to modify the directories in this file to match your local directories :

```
25 MODEL_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Models
26 TYPE_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Types
27 FUNCTION_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Functions
28 CONCEPT_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Concepts
29 MESSAGE_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/MessagesAndContainers/Fixed/Messages
30 CONTAINER_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/MessagesAndContainers/Fixed/Containers
31 MODEL_EXEC_LOG=false
32 JarPaths=/tmp/OnLEPClusterInstall
33 CONFIG_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Configs
34 nodeId=1
```

- EngineConf.json – modify the internal values to match your local environment (see Appendix for a an example of a full file). This file is used in the OnLEP engine.

  For example, to configure the "Node 1" change the values (shown in red) for these JSON properties:

```
"NodeId": "1",
"NodePort": 6541,
"NodeIpAddr": "localhost",
"JarPaths": [
"/tmp/OnLEPInstall"
],
"Scala_home": "<path>/scala-2.10.4",
"Java_home": "<path>/jdk1.8.0_05",
"Classpath": ".:/tmp/OnLEPInstall/metadata_2.10-1.0.jar:/tm…..
```

4. Run the Metadata.API application by issuing the following command:

   <installDirectory>/metaDataAPI-1.0 –config <configDirectory>MetadataAPIConfig.properties

   You will see the UI shown here.

5. First, use option 33 in MetaData.API to load *EngineConf.json* into MetaData management database. You would have modified this in the configuration step.

```
25 MODEL_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Models
26 TYPE_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Types
27 FUNCTION_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Functions
28 CONCEPT_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Concepts
29 MESSAGE_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/MessagesAndContainers/Fixed/Messages
30 CONTAINER_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/MessagesAndContainers/Fixed/Containers
31 MODEL_EXEC_LOG=false
32 JarPaths=/tmp/OnLEPClusterInstall
33 CONFIG_FILES_DIR=/home/rich/github/Med/RTD/trunk/SampleApplication/Medical/Configs
34 nodeId=1
```

6. Create the models and elements to run the engine. Enter the number of the item you want to create. It is fairly intuitive from there.

7. Using the Metadata.API , add metadata objects in the following order:

1. Messages
    - hl7.json
    - inpatientclaim.json
    - outpatientclaim.json
    - beneficiary.json

2. Containers
    - idCodeDim.json
    - CoughCodes.json
    - EnvCodes.json
    - SmokeCodes.json
    - SputumCodes.json
    - DyspnoeaCodes.json
    - hcpcsCodes.json
    - icd9DiagnosisCodes.json
    - icd9ProcedureCodes.json
    - dgRelGrpCodes.json
    - lneProcIndTable.json
    - provNumTable.json

3. Models
    - COPDv1.xml

4. Prepare the kvstores using the new metadata-generated jars:

    Run
    <localCloneDirectory>/RTD_ReadOnly/trunk/SampleApplication/Medical/bin/PrepKvStores.sh
    <installPath> <sourcePath> <engineConfigFile>

Windows:

Note: the script for KVStore initialization will be included in a future release.

Make sure Scala and sbt paths are in the PATH environmental variable.

Example: yours may look different:

C:\ProgramData\Oracle\Java\javapath;C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Program Files (x86)\Common Files\Microsoft Shared\Windows Live;%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Live\Shared;;**C:\Program Files (x86)\scala\bin;C:\Program Files (x86)\sbt\\bin**

Install the demo you wish to use:

For instance, execute 'installOnLEP_Medical.bat'
Note: Compression to .gz is commented out in the bat file and needs to be done manually, then copied over manually.

Running OnLEP and processing data

1. Run the engine:
    a. Execute:
    /path/to/install/loc/OnLEPManager-1.0 --config </path/to/config/file>

    Example: /tmp/OnLEPInstall/OnLEPManager-1.0 --config /tmp/OnLEPInstall/COPD.properties

    b. You can see the input file here:
        /path/to/install/location/msgdata/messages_new_format_all.csv.gz

    Then examine the following files to see the output.

        /path/to/install/location/logs/output1.txt.gz

    Quit the engine by typing "quit". Ctrl-C should also work.

Multi-Node Account Setup Instructions

Each OnLEP cluster installed on a network is ideally run by a distinct user account. These instructions provide the basics for setting up such a user so that it can easily manage the cluster administration.

A user name and group should be selected to be the owner and user that runs the cluster. It can be any useful and meaningful name not already in use. In the examples below, the name *onlep* is used for both the account and group name. Two nodes named *ls19.dc.npario.com* and *ls20.dc.npario.com* are used to illustrate the setup on two machines.

Admin setup items – verify or do the following:

1. If the *onlep* group does not exist on the prospective cluster nodes, add it to each node with the following command:

   sudo groupadd onlep

2. Establish a user account on each prospective cluster node making the new user use the onlep group as its default login group. Be sure to establish a home directory for the user. Example:

   sudo useradd --gid onlep --create-home onlep

3. Set the password for the onlep cluster login to something using sudo

   sudo passwd onlep

This assumes that you can get to these machine from another account and have sudoer privileges.

4. If the onlep release is staged from source on one of the machines in the new cluster or cluster's network, make sure that the staging user (or the developer) account is also a member of the *onlep* group. This can be done with this command:

   sudo usermod -G onlep releaseAdmin

where the user account is releaseAdmin. This really is only needed on this one machine, not every node in the cluster, although THAT would not necessarily be a bad idea. This is security policy dependent.

5. Passphrase-less access to/from all participating nodes in the *onlep* cluster required for proper operation. Each node could have its own public and private key if your security policy requires it. However, it is far easier to login to one of the nodes as the onlep user and set up the .ssh keys once,

then propagate the entire directory to the other nodes in the cluster. Assuming the accounts are new, there is really no harm in replacing the entire .ssh directory. If you have existing keys there, however, don't replace. Instead append.

To append to the .ssh keys

1. generate a public private key

    ssh-keygen

It will prompt for a key type. Take the default (RSA 2048) by hitting return. Then it will ask for a passphrase; hit return for "passphraseless" behavior. It will prompt again to type the same passphrase; once again hit return. This will generate a public and private key for the current user account.

2. Install the public key on the current machine.

To install the new public key in the current directory, you can cat the public key into a file called .ssh/authorized_keys or perhaps a little easier use this approach, as it will properly set the access permissions for the fil
e it will create. If the current machine was named ls19.dc.npario.com and the onlep account being setup is onlep, do the following:

    ssh-copy-id onlep@ls19.dc.npario.com

You will have to type the password this one time to establish the authorized key file, but after that you should be able to access without giving a passphrase.

3. Verify that the $HOME/.ssh permissions is strict. There should be no permitted access to the .ssh directory except by the owner.

4. As the onlep account, Zip/Bzip/Gzip the .ssh folder in its entirety and leave it in $HOME. For example:

    tar cvjf onlepNodeSSH.bz2 .ssh

This creates an archive that will be copied around to the other nodes that are to participate in the cluster.

5. As the onlep account, copy the archive to every node in the cluster and unzip it. For example, if ls20.dc.npario.com were another node in the cluster to be setup, do the following:

    scp onlepNodeSSH.bz2 ls20.dc.npario.com:

This will leave the tarball in the onlep user's ls20 home folder. Do this same action for all the nodes to particpate or might participate in the cluster being set up.

6. ssh to each node as the onlep user and decompress the file copied in the previous step. For example,

    tar xvjf onlepNodeSSH.bz2

This will update the .ssh folder with the authorized_keys as well as the public and private keys that were generated in part a) above. To verify that it worked, try to ssh to the original machine where the keys were generated. For example,

    ssh ls19.dc.npario.com

If successful, then you should login immediately (not requiring a passphrase from the console).

7. Check further. A good check would be to login to one of the cluster nodes to particpate in the onlep cluster being formed AS the onlep account and try to log in to every other machine in the cluster. These logins should succeed without a passphrase being required.

Should there be problems, there are many resources on the web that describe the setup of a passphrase-less account. In particular, the access permissions for the account and its .ssh need to be strict (no group or global permissions should be allowed). Ssh will complain if these requirements are not met.

8. Finally, remove the tarball that has been passed around to all of the nodes from each onlep user account on each node.

More information. The instructions above are indicative, not authoritative, of what must be done to establish password-less access. If you experience problems, consult the documentation for your Linux enlistment for special requirements that your environment may require.

Installing OnLEP to a cluster from your Git Repository

OnLEP is typically installed from the trunk directory of an “install” or “release” machine, building the OnLEP installation directory on the local system, creating a tarball of that directory, sending it to the nodes that are part of the OnLEP cluster, untar and decompress the tarball there and move the resulting directory into place.

Several scripts are part of the repository that are used to accomplish this. These scripts must be moved onto your PATH and given “execute” permissions. They are:

1. trunk/SampleApplication/Medical/bin/clusterInstallOnLEP.sh

2. trunk/SampleApplication/Medical/bin/installOnLEP_Medical.sh
3. trunk/SampleApplication/Medical/bin/nodeInfoExtract.sh
4. trunk/Pmml/Scripts/sbtProjDependencies/src/main/scala/sbtProjDependencies.scala

The ***clusterInstallOnLEP.sh*** *script* controls the installation, using the cluster configuration information either found in the supplied NodeConfigPath argument or from the metadata store described in the MetadataAPIConfig file. In the first case, the metadata store may have no configuration and the supplied NodeConfigPath will provide it. In the second case, where no NodeConfigPath is supplied, the metadata store is expected to have one. Should a valid configuration not be found, the script will issue a RuntimeException with a useful message describing the particular issue. Here are examples of script invocation

clusterInstallOnLEP.sh
--MetadataAPIConfig SampleApplication/Medical/Configs/MetadataAPIConfig.properties

clusterInstallOnLEP.sh
--NodeConfigPath SampleApplication/Medical/Configs/Engine2BoxConfigV1.json

The ***installOnLEP_Medical.sh*** will build the OnLEP install directory. The *nodeInfoExtract.*sh is used to determine which cluster nodes are to receive the build. The s*btProjDependencies* is a useful tool used by a number of scripts that are part of OnLEP. In this case, it is used to build the classpath for the *nodeInfoExtract* `application.

The OnLEP cluster start script - **startOnLEPCluster.sh**, like the install script, can take one or two arguments. If the NodeConfigPath is present, its cluster config will be added to the metadata store specified with the MetadataAPIConfig argument. If not present, the metadata store is expected to have a configuration. Should a valid configuration not be found, the script will issue a RuntimeException with a useful message describing the particular issue.

startOnLEPCluster.sh
--MetadataAPIConfig <metadataAPICfgPath> [--NodeConfigPath <onlepCfgPath>]

Invoke the installer

Use the following command by invoking it from the trunk directory of the local Git repository which is to be installed. Run it from the account that is to be used to run/manage the OnLEP cluster and refer to directories that are write-able by that account.

```
Cluster InstallOnLEP.sh
  --MetadataAPIConfig  SampleApplication/Medical/Configs/MetadataAPIConfig.properties
  --NodeConfigPath SampleApplication/Medical/Configs/Engine2BoxConfigV1.json
```

Currently two configuration files are required, one for the MetadataAPI and the other that describes at least the cluster for the OnLEPManager. The cluster description is the key information as to what sort of

distribution and what machines and nodes are involved in the installation. Note that multiple nodes can if desired be installed on the same physical computer.

For more information about what these configuration files contain, see <some reference to other documentation describing the configuration>.

Appendix

Please use this example of the EngineConf.Json file as a guide to the values you need to change in the system you are using to run the OnLEP engine. Gray shading indicates those values.


```
{
  "Clusters": [
    {
      "ClusterId": "ligadata1",
      "Config": {
        "DataStore": "{\"StoreType\": \"cassandra\",\"SchemaName\": \"testdata\",\"Location\":
\"localhost\"}",
        "StatusInfo": "{\"StoreType\": \"cassandra\",\"SchemaName\":
\"statusinfo\",\"Location\": \"localhost\"}",
        "ZooKeeperInfo": "{\"ZooKeeperNodeBasePath\": \"/ligadata\",\"ZooKeeperConnectString\":
\"localhost:2181\",\"ZooKeeperSessionTimeoutMs\": \"250\",\"ZooKeeperConnectionTimeoutMs\":
\"30000\"}",
        "EnvironmentContext":
"{\"classname\":\"com.ligadata.SimpleEnvContextImpl.SimpleEnvContextImpl$\",\"jarname\":\"simple
envcontextimpl_2.10-1.0.jar\",\"dependencyjars\":[\"log4j-1.2.17.jar\",\"onlepbase_2.10-
1.0.jar\",\"metadata_2.10-1.0.jar\",\"serialize_2.10-1.0.jar\",\"storage_2.10-
0.0.0.2.jar\",\"asm-3.1.jar\",\"metrics-core-3.0.2.jar\",\"cassandra-driver-core-
2.0.2.jar\",\"kryo-2.21.jar\",\"minlog-1.2.jar\",\"reflectasm-1.07-shaded.jar\",\"jackson-
annotations-2.3.0.jar\",\"jackson-core-2.3.1.jar\",\"jackson-databind-2.3.1.jar\",\"findbugs-
annotations-1.3.9-1.jar\",\"jsr305-1.3.9.jar\",\"google-collections-1.0.jar\",\"guava-
16.0.1.jar\",\"protobuf-java-2.5.0.jar\",\"protobuf-java-2.6.0.jar\",\"java-xmlbuilder-
0.4.jar\",\"jsch-0.1.42.jar\",\"compress-lzf-0.9.1.jar\",\"junit-interface-0.11-RC1.jar\",\"je-
4.0.92.jar\",\"jersey-core-1.9.jar\",\"jersey-json-1.9.jar\",\"jersey-server-1.9.jar\",\"jaxb-
impl-2.2.3-1.jar\",\"paranamer-2.3.jar\",\"paranamer-2.6.jar\",\"chill-java-
0.3.6.jar\",\"chill_2.10-0.3.6.jar\",\"commons-beanutils-core-1.8.0.jar\",\"commons-beanutils-
1.7.0.jar\",\"commons-cli-1.2.jar\",\"commons-codec-1.9.jar\",\"commons-collections-
3.2.1.jar\",\"commons-configuration-1.6.jar\",\"commons-dbcp-1.2.2.jar\",\"commons-digester-
1.8.jar\",\"commons-el-1.0.jar\",\"commons-httpclient-3.1.jar\",\"commons-io-
2.4.jar\",\"commons-lang-2.6.jar\",\"commons-logging-1.1.3.jar\",\"commons-net-
3.1.jar\",\"commons-pool-1.5.2.jar\",\"netty-3.9.0.Final.jar\",\"activation-1.1.jar\",\"jsp-api-
2.1.jar\",\"servlet-api-2.5.jar\",\"jaxb-api-2.2.2.jar\",\"stax-api-1.0-2.jar\",\"jline-
0.9.94.jar\",\"joda-time-2.3.jar\",\"junit-4.11.jar\",\"log4j-1.2.17.jar\",\"jets3t-
0.9.0.jar\",\"jna-3.2.7.jar\",\"avro-1.7.4.jar\",\"commons-compress-1.4.1.jar\",\"commons-math3-
3.1.1.jar\",\"hadoop-annotations-2.4.1.jar\",\"hadoop-auth-2.4.1.jar\",\"hadoop-common-
2.4.1.jar\",\"hbase-client-0.98.4-hadoop2.jar\",\"hbase-common-0.98.4-hadoop2.jar\",\"hbase-
protocol-0.98.4-hadoop2.jar\",\"httpclient-4.2.5.jar\",\"httpcore-4.2.4.jar\",\"zookeeper-
3.4.6.jar\",\"htrace-core-2.04.jar\",\"jackson-core-asl-1.8.8.jar\",\"jackson-jaxrs-
1.8.3.jar\",\"jackson-mapper-asl-1.8.8.jar\",\"jackson-xc-1.8.3.jar\",\"jettison-
1.1.jar\",\"hamcrest-core-1.3.jar\",\"jdom-1.1.jar\",\"joda-convert-1.6.jar\",\"json4s-ast_2.10-
3.2.9.jar\",\"json4s-core_2.10-3.2.9.jar\",\"json4s-jackson_2.10-3.2.9.jar\",\"json4s-
native_2.10-3.2.9.jar\",\"mapdb-1.0.6.jar\",\"jetty-util-6.1.26.jar\",\"jetty-
6.1.26.jar\",\"objenesis-1.2.jar\",\"asm-commons-4.0.jar\",\"asm-tree-4.0.jar\",\"asm-
4.0.jar\",\"scalap-2.10.0.jar\",\"test-interface-1.0.jar\",\"quasiquotes_2.10.4-2.0.0-
M6.jar\",\"scalatest_2.10-2.2.0.jar\",\"slf4j-api-1.7.7.jar\",\"slf4j-log4j12-1.7.5.jar\",\"xz-
1.0.jar\",\"snappy-java-1.0.4.1.jar\",\"jasper-compiler-5.5.23.jar\",\"jasper-runtime-
5.5.23.jar\",\"voldemort-0.96.jar\",\"xmlenc-0.52.jar\"]}"
      },
      "Nodes": [
        {
          "NodeId": "1",
          "NodePort": 6541,
          "NodeIpAddr": "localhost",
          "JarPaths": [
```

```
            "/tmp/OnLEPInstall"
          ],
          "Scala_home": "/home/xxxxxx/scala-2.10.4",
          "Java_home": "/home/xxxxxx/jdk1.8.0_05",
          "Classpath": ".:/tmp/OnLEPInstall/metadata_2.10-
1.0.jar:/tmp/OnLEPInstall/basefunctions_2.10-0.1.0.jar:/tmp/OnLEPInstall/messagedef_2.10-
1.0.jar:/tmp/OnLEPInstall/methodextractor_2.10-1.0.jar:/tmp/OnLEPInstall/pmmlcompiler_2.10-
1.0.jar:/tmp/OnLEPInstall/onlepbase_2.10-1.0.jar:/tmp/OnLEPInstall/bootstrap_2.10-
1.0.jar:/tmp/OnLEPInstall/joda-time-2.3.jar:/tmp/OnLEPInstall/joda-convert-
1.6.jar:/tmp/OnLEPInstall/basetypes_2.10-0.1.0.jar:/tmp/OnLEPInstall/pmmludfs_2.10-
1.0.jar:/tmp/OnLEPInstall/pmmlruntime_2.10-1.0.jar:/tmp/OnLEPInstall/json4s-native_2.10-
3.2.9.jar:/tmp/OnLEPInstall/json4s-core_2.10-3.2.9.jar:/tmp/OnLEPInstall/json4s-ast_2.10-
3.2.9.jar:/tmp/OnLEPInstall/jackson-databind-2.3.1.jar:/tmp/OnLEPInstall/jackson-annotations-
2.3.0.jar:/tmp/OnLEPInstall/json4s-jackson_2.10-3.2.9.jar:/tmp/OnLEPInstall/jackson-core-
2.3.1.jar:/tmp/OnLEPInstall/log4j-1.2.17.jar"
        },
        {
          "NodeId": "2",
          "NodePort": 6542,
          "NodeIpAddr": "localhost",
          "JarPaths": [
            "/tmp/OnLEPInstall"
          ],
          "Scala_home": "/home/xxxxxx/scala-2.10.4",
          "Java_home": "/home/xxxxxx/jdk1.8.0_05",
          "Classpath": ".:/tmp/OnLEPInstall/metadata_2.10-
1.0.jar:/tmp/OnLEPInstall/basefunctions_2.10-0.1.0.jar:/tmp/OnLEPInstall/messagedef_2.10-
1.0.jar:/tmp/OnLEPInstall/methodextractor_2.10-1.0.jar:/tmp/OnLEPInstall/pmmlcompiler_2.10-
1.0.jar:/tmp/OnLEPInstall/onlepbase_2.10-1.0.jar:/tmp/OnLEPInstall/bootstrap_2.10-
1.0.jar:/tmp/OnLEPInstall/joda-time-2.3.jar:/tmp/OnLEPInstall/joda-convert-
1.6.jar:/tmp/OnLEPInstall/basetypes_2.10-0.1.0.jar:/tmp/OnLEPInstall/pmmludfs_2.10-
1.0.jar:/tmp/OnLEPInstall/pmmlruntime_2.10-1.0.jar:/tmp/OnLEPInstall/json4s-native_2.10-
3.2.9.jar:/tmp/OnLEPInstall/json4s-core_2.10-3.2.9.jar:/tmp/OnLEPInstall/json4s-ast_2.10-
3.2.9.jar:/tmp/OnLEPInstall/jackson-databind-2.3.1.jar:/tmp/OnLEPInstall/jackson-annotations-
2.3.0.jar:/tmp/OnLEPInstall/json4s-jackson_2.10-3.2.9.jar:/tmp/OnLEPInstall/jackson-core-
2.3.1.jar:/tmp/OnLEPInstall/log4j-1.2.17.jar"
        },
        {
          "NodeId": "4",
          "NodePort": 6544,
          "NodeIpAddr": "localhost",
          "JarPaths": [
            "/tmp/OnLEPInstall"
          ],
          "Scala_home": "/home/xxxxxx/scala-2.10.4",
          "Java_home": "/home/xxxxxx/jdk1.8.0_05",
          "Classpath": ".:/tmp/OnLEPInstall/metadata_2.10-
1.0.jar:/tmp/OnLEPInstall/basefunctions_2.10-0.1.0.jar:/tmp/OnLEPInstall/messagedef_2.10-
1.0.jar:/tmp/OnLEPInstall/methodextractor_2.10-1.0.jar:/tmp/OnLEPInstall/pmmlcompiler_2.10-
1.0.jar:/tmp/OnLEPInstall/onlepbase_2.10-1.0.jar:/tmp/OnLEPInstall/bootstrap_2.10-
1.0.jar:/tmp/OnLEPInstall/joda-time-2.3.jar:/tmp/OnLEPInstall/joda-convert-
1.6.jar:/tmp/OnLEPInstall/basetypes_2.10-0.1.0.jar:/tmp/OnLEPInstall/pmmludfs_2.10-
1.0.jar:/tmp/OnLEPInstall/pmmlruntime_2.10-1.0.jar:/tmp/OnLEPInstall/json4s-native_2.10-
3.2.9.jar:/tmp/OnLEPInstall/json4s-core_2.10-3.2.9.jar:/tmp/OnLEPInstall/json4s-ast_2.10-
3.2.9.jar:/tmp/OnLEPInstall/jackson-databind-2.3.1.jar:/tmp/OnLEPInstall/jackson-annotations-
2.3.0.jar:/tmp/OnLEPInstall/json4s-jackson_2.10-3.2.9.jar:/tmp/OnLEPInstall/jackson-core-
2.3.1.jar:/tmp/OnLEPInstall/log4j-1.2.17.jar"
        }
      ]
    }
```

```
  ],
  "Adapters": [
    {
      "Name": "TestIn_1",
      "TypeString": "Input",
      "DataFormat": "CSV",
      "ClassName": "com.ligadata.InputAdapters.KafkaSimpleConsumer$",
      "JarName": "kafkasimpleinputoutputadapters_2.10-1.0.jar",
      "DependencyJars": [
        "jopt-simple-3.2.jar",
        "kafka_2.10-0.8.1.1.jar",
        "log4j-1.2.15.jar",
        "metrics-core-2.2.0.jar",
        "slf4j-api-1.7.2.jar",
        "snappy-java-1.0.5.jar",
        "zkclient-0.3.jar",
        "zookeeper-3.3.4.jar",
        "onlepbase_2.10-1.0.jar"
      ],
      "AdapterSpecificCfg": "{\"HostList\": \"localhost:9092\",\"TopicName\": \"testin_1\" }"
    },
    {
      "Name": "TestOut_In_1",
      "TypeString": "Validate",
      "DataFormat": "JSON",
      "ClassName": "com.ligadata.InputAdapters.KafkaSimpleConsumer$",
      "JarName": "kafkasimpleinputoutputadapters_2.10-1.0.jar",
      "DependencyJars": [
        "jopt-simple-3.2.jar",
        "kafka_2.10-0.8.1.1.jar",
        "log4j-1.2.15.jar",
        "metrics-core-2.2.0.jar",
        "slf4j-api-1.7.2.jar",
        "snappy-java-1.0.5.jar",
        "zkclient-0.3.jar",
        "zookeeper-3.3.4.jar",
        "onlepbase_2.10-1.0.jar"
      ],
      "AdapterSpecificCfg": "{\"HostList\": \"localhost:9092\",\"TopicName\": \"testout_1\" }"
    },
    {
      "Name": "TestOut_1",
      "TypeString": "Output",
      "InputAdapterToVerify": "TestOut_In_1",
      "ClassName": "com.ligadata.OutputAdapters.KafkaProducer$",
      "JarName": "kafkasimpleinputoutputadapters_2.10-1.0.jar",
      "DependencyJars": [
        "jopt-simple-3.2.jar",
        "kafka_2.10-0.8.1.1.jar",
        "log4j-1.2.15.jar",
        "metrics-core-2.2.0.jar",
        "slf4j-api-1.7.2.jar",
        "snappy-java-1.0.5.jar",
        "zkclient-0.3.jar",
        "zookeeper-3.3.4.jar",
        "onlepbase_2.10-1.0.jar"
      ],
      "AdapterSpecificCfg": "{\"HostList\": \"localhost:9092\",\"TopicName\": \"testout_1\" }"
    },
    {
```

```
      "Name": "TestStatus_1",
      "TypeString": "Status",
      "ClassName": "com.ligadata.OutputAdapters.KafkaProducer$",
      "JarName": "kafkasimpleinputoutputadapters_2.10-1.0.jar",
      "DependencyJars": [
        "jopt-simple-3.2.jar",
        "kafka_2.10-0.8.1.1.jar",
        "log4j-1.2.15.jar",
        "metrics-core-2.2.0.jar",
        "slf4j-api-1.7.2.jar",
        "snappy-java-1.0.5.jar",
        "zkclient-0.3.jar",
        "zookeeper-3.3.4.jar",
        "onlepbase_2.10-1.0.jar"
      ],
      "AdapterSpecificCfg": "{\"HostList\": \"localhost:9092\",\"TopicName\": \"teststatus_1\"
}"
    }
  ]
}
```