LoT Analytics Engine
User Guide

This guide helps in setting up and configuring the Analytics Engine System. Following are the main components of the system which must be deployed and configured as described:

1. Web Services

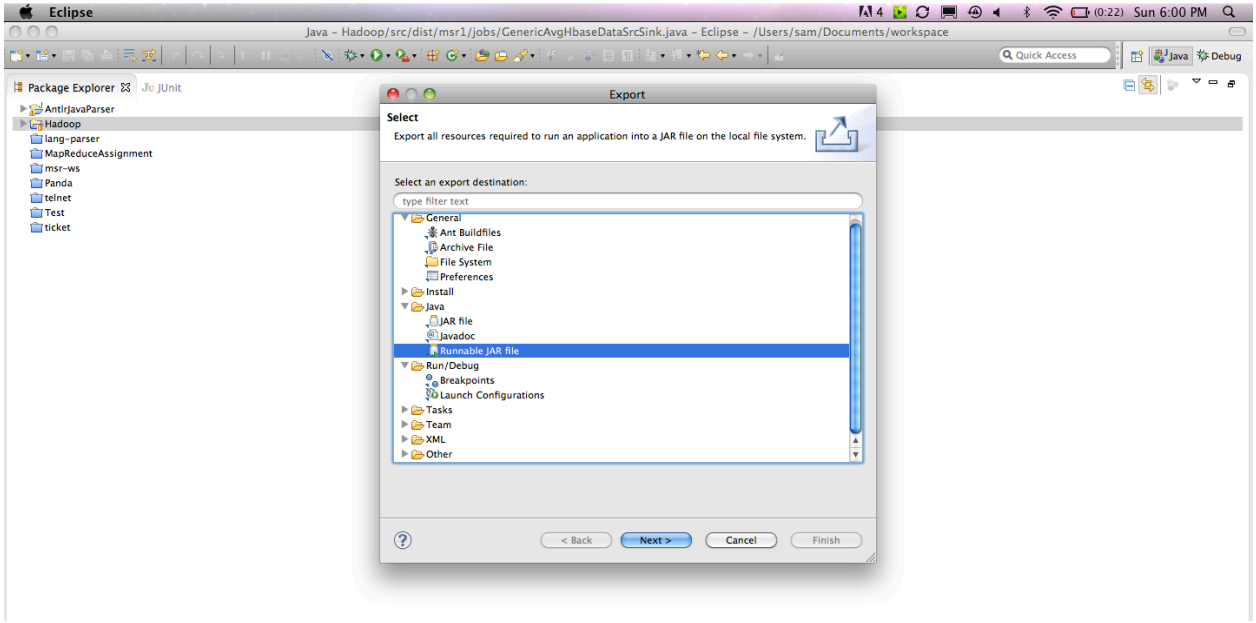
1. Web services has been packaged inside project LoTAnalyticsEngine in codeplex.
2. In order to compile the source code of the Web Services you need to have installed Apache **Maven** in your system.
3. Since it is a maven project, all dependencies, configured in pom.xml, could be resolved my **mvn compile.** Command is run from root directory of the project.
4. This project could be **packaged as war** using command: **mvn clean install**
   This creates msr-ws.war file in target folder of the project.
5. This war can be deployed on tomcat application server.

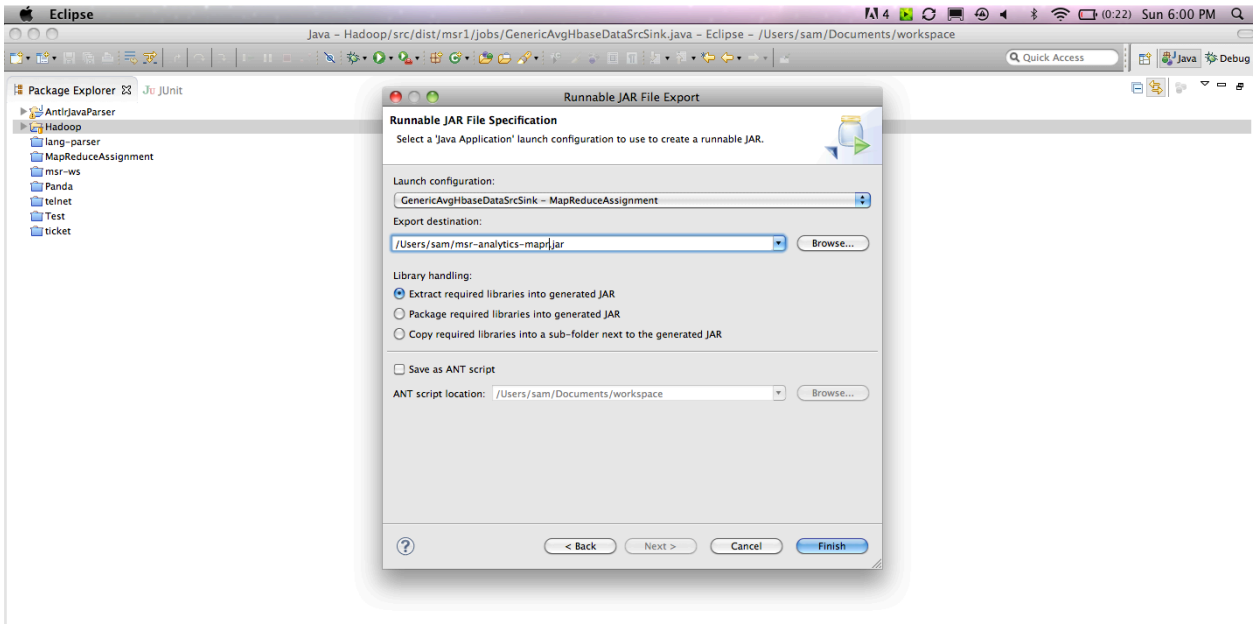
   For development purpose you can also create, run and deploy the war (skipping steps 4-5) using the embedded tomcat (which comes with our system) simply by running following command: **mvn clean tomcat6:run** from project root directory. This runs the tomcat on port 9999 by default (port could be changed in pom.xml), on localhost. You can access the Application at localhost:9999/msr-ws.

2. Hadoop Analytics

Hadoop Job is responsible for batch aggregation of data. This data is collected from three possible sources: real time Data from LoT, research data and data generated by our custom DataSimultor discussed in Section 3 Analysis and Design.

Package as Jar: The Hadoop Project (could be found in root of our project in CodePlex), must be exported as ‘runnable jar’ (say from Eclipse as shown in figure below).





HBase is used as data source and data sink for our Hadoop Job. Same job could be used to aggregate data in different ways based on level and the timescale. These two arguments are passed to the Hadoop Job when it is run:

```
hadoop jar msr-analytics-mapr.jar msr1.GenericAvgHbaseDataSrcSink  <level> <timescale>
```

This command calculates average of collected data over given category and given timescale. If data needs to be aggregated in more than one format, multiple Hadoop schedulers must be setup which different configuration for levels and time-scales.

For example:

1. hadoop jar mrs-nb1.jar msr1.GenericAvgHbaseDataSrcSink research/group/source/location daily
2. hadoop jar mrs-nb1.jar msr1.GenericAvgHbaseDataSrcSink research////location daily

Command-1 computes daily average reading for each research/group/source/location combination.

While Command-2 computes daily average for each unique research/location combination, that is, it computes daily average for each research and locations in that research.

Levels:

Levels are input to the Hadoop job in form: research/group/source/*category*

where *category* could be: deviceType, location, deviceId

Each of these properties are recorded from the data-source (LoT, IoT, Simulator):

- research: Research ID e.g. UCL
  Research IDs we have used in our scenarios:
  `"Ucl", "MS-R1", "MS-R2", "MS-R3"`

- group: Group ID e.g. home, hospital
  Group IDs we have used in our scenarios:
  `“home”, “hospital”`

- source: ID of house/place where LoT is deployed e.g. 12343
  Source IDs we have used in our scenarios:
  `“12300”, "12311", "12322", "12333", "12344”`

- leaf
  - deviceType: Each device falls into a type like lights, cooling, heating and others
    DeviceTypes we have used in our scenarios:
    `"lights", "cooling", "kitchen-appliances", "refrigerator", "heating", "entertainment", "temperature", "totalec", "door" , "heart-rate-mtr"`

  - location: location where device is placed like kitchen, living-room.
    Different locations we have used in our scenarios:
    `"Miami", "London", "Leicester"`

  - deviceId: Unique id for each device e.g. E01, v00, f02. DeviceId starts with letter ‘e’, ‘v’, ‘f’ which signifies the type of reading this device records like e: electricity consumption, v: vacuum, f: frequency, t: temperature.
    DeviceIDs we have used in our scenarios:
    `"e01", "e02", "e03", "e04", "e05","e06", "deg000", "e0000", "f01", "hrm00","deg1", "deg1", "deg3"`

Analytic Engine is actually independent of these properties in levels, these are configured only at Simulator and Web Portal to provide options for the users to select range of data on which they want to apply analytics to.

Following are few possible levels that system:

Possible levels	Description	Useful for Queries
research/group/source/deviceType	Computes average reading for each batch based on research/group/source/deviceType	Compare electricity usage for all types of devices in house ‘x’, group ‘y ‘and research ‘z’.
research/group/source/deviceId	Computes average reading for each batch based on research/group/source/deviceId	Compare how much each device contributes to in overall electricity consumption in house ‘x’, group ‘y’ and research ‘z’.
research/group/source/	Computes average for all devices in each research/group/source	Get top five houses which were the highest consumer of electricity last year
research///location	Computes average for all locations in each research	In given research ‘x’, get location which is the highest consumer of electricity
research///deviceType	Computes average for all device-types in each research	In given research ‘x’, get device-type which is the highest consumer of electricity
Research///	Computes average over all average for all researches	Compare average electricity consumption over past 5 years for research ‘x’ and research ‘y’

Time-scale:
Time-scale is given as input to the hadoop job. Following are the possible values for time-scale:

Possible time-scale	Description	Useful for Queries
Hourly	Aggregates data for every hour	Get time when there was highest average consumption of electricity during the day
Daily	Aggregates data for each day	Get top 5 days last year when there was highest average

		consumption of electricity
Monthly	Aggregates data for each month	Get month during which there is highest average consumption of electricity every year
Annually	Aggregates data for each year	Compare electricity consumption for last 10 years

Levels/Timescales **configurations used for our scenarios** with test data:

1. hadoop jar mrs-lot-analytics.jar msr1.GenericAvgHbaseDataSrcSink research/group/source/location hourly
2. hadoop jar mrs-lot-analytics.jar msr1.GenericAvgHbaseDataSrcSink research///location hourly
3. hadoop jar mrs-lot-analytics.jar msr1.GenericAvgHbaseDataSrcSink research/group/ /deviceType hourly

3. Data Stores

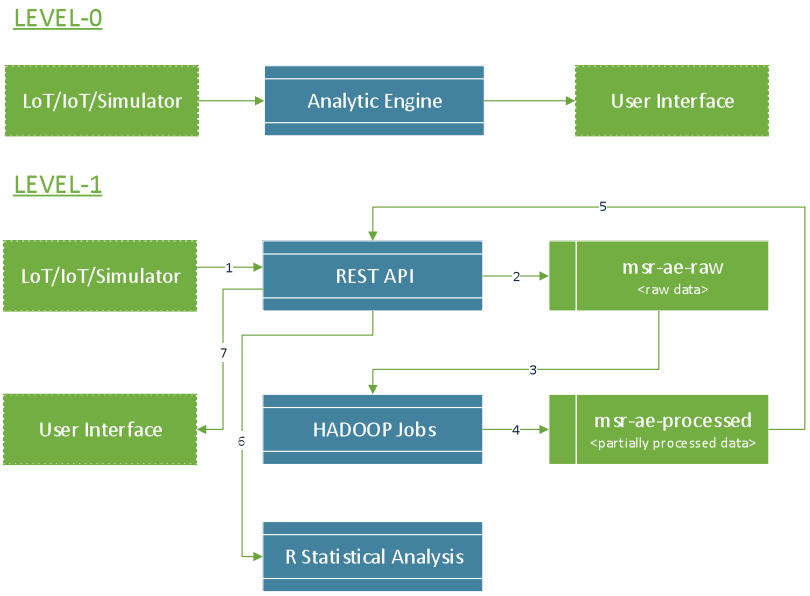
We are using following datastores to store raw data (from HomeOS/LoT, input to Hadoop) and processed (aggregated) data (output from Hadoop) respectively:

msr-lot-raw
Following command must be run on Hbase to create this datatore:
create ‘msr-lot-raw’, ‘details’

msr-lot-processed
Following command must be run on Hbase to create this datatore:

create ‘msr-lot-processed’, ‘hourly’, ‘daily’, ‘monthly’, ‘annually’

Following figures shows the data flow throw these datastores:



3. R (Rserve)

Following are the configurations required:

1. Go to http://www.stats.bris.ac.uk/R/ and Download R
2. Install R
3. Run R as administer
4. To download Rserve library, you need to type a command, "install.packages("Rserve")".

Integration of R with WebServices

5. Download RserveEngine.jar and REngine.jar
6. In Add those external jar into msr-ws project

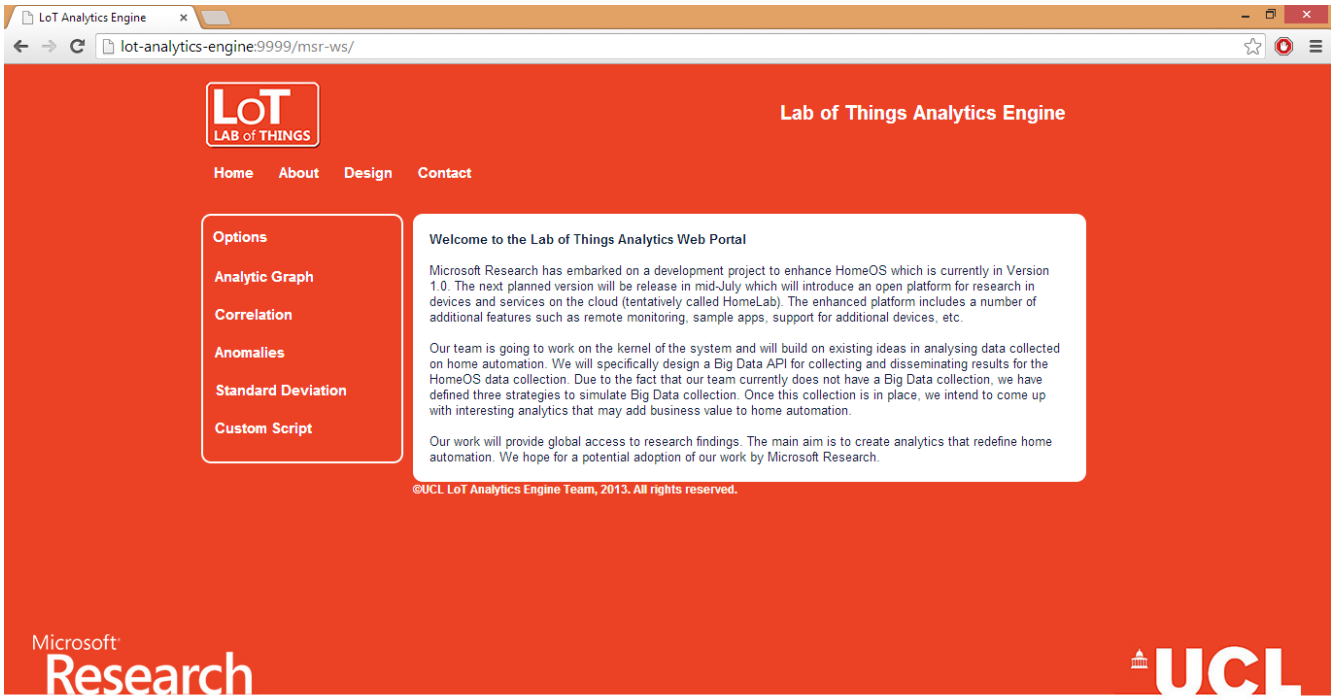
To run R:

7. Run R as administrator
8. Type a command "library(Rserve)" and then "Rserve()".
9. Then Rserve will be running on background

4. Web Portal

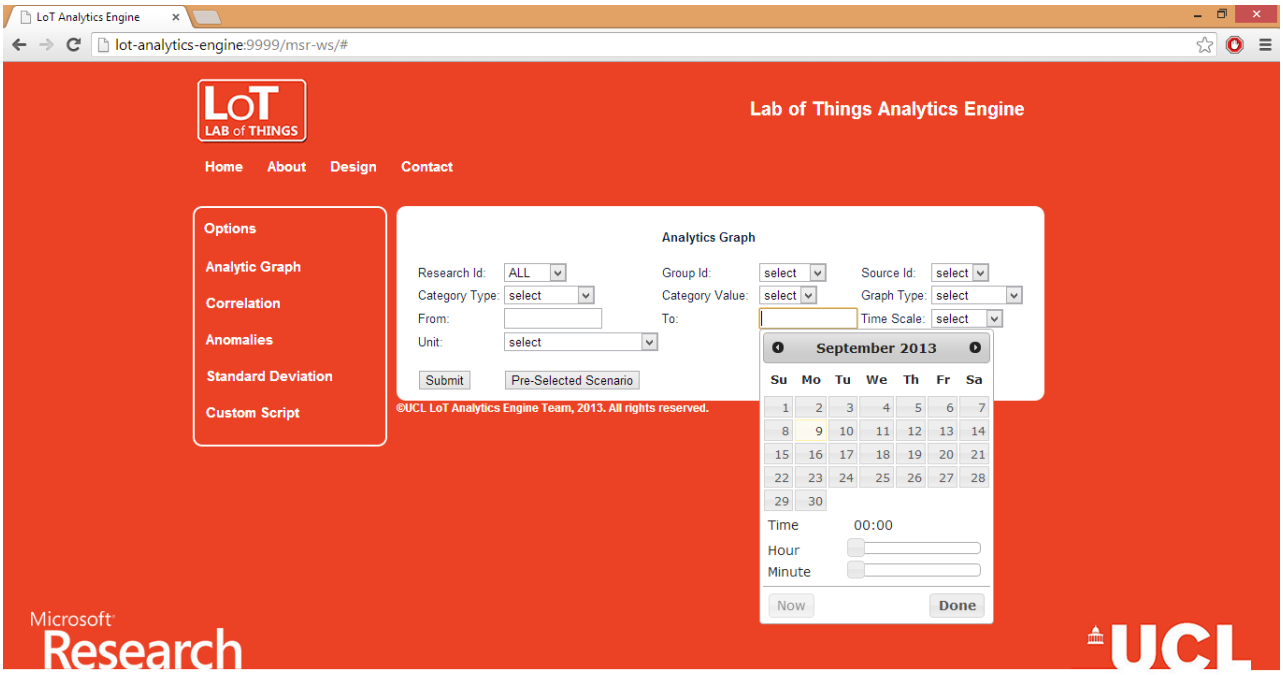
1. To access the web portal the war file mentioned in the web services section of this user guide must be deployed on a server.
2. The url for the web portal is <your server:port>/msr-ws for example if you are running the web services on your localhost on port 9999 then the url to access the web portal is localhost:9999/msr-ws (please refer to section 1 to see how to package/deploy this project)

3. The web portal has various pre-configured scenarios, which are accessible through the button marked “Pre Selected Scenarios”.
4. The webportal is located in the webapp directory of the web services source code.
5. All the javascript,jquery and html5 code needed for the webportal are included in the mainPage.jsp
6. Style elements are located in style.css and the images in the images folder.
7. The web portal will function correctly only if all instructions mentioned in the R and web services sections are followed.
8. Below is a screenshot of the web portal. The options on the left are the statistical functions we have provided.

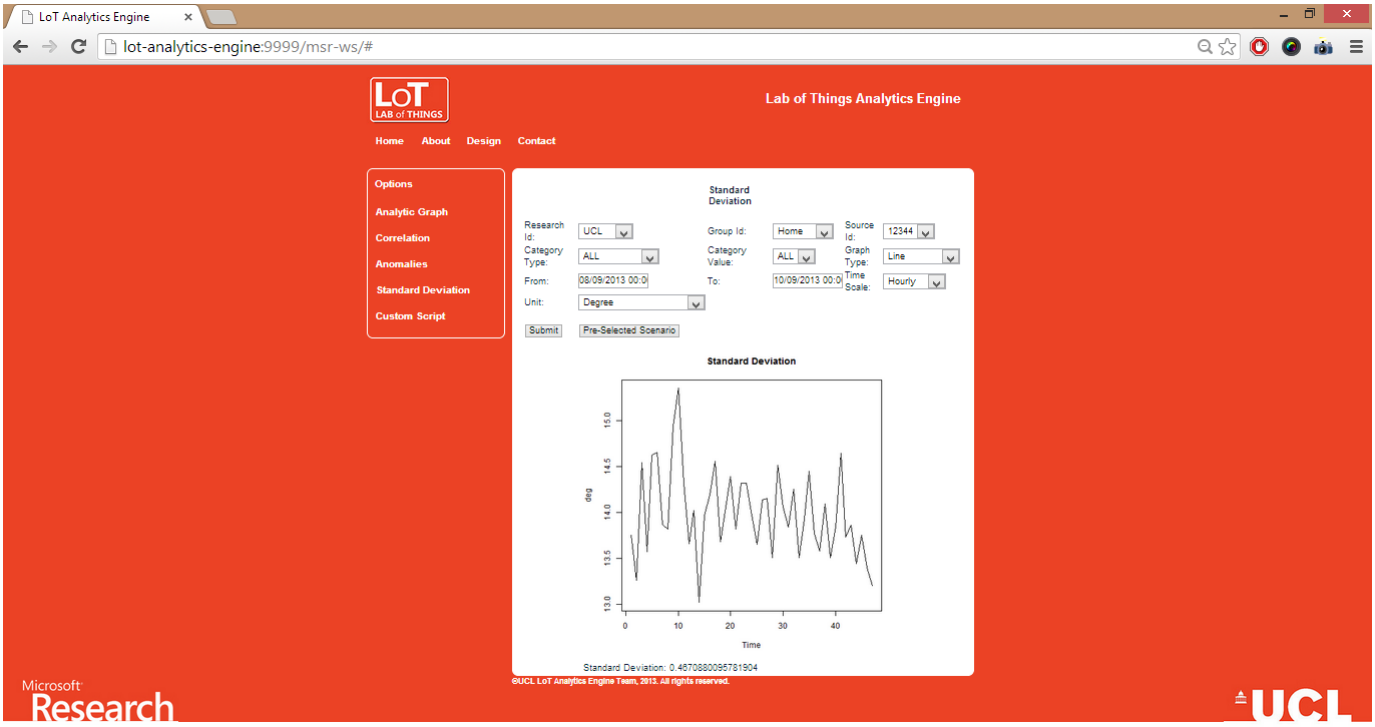


10. Clicking on each option brings up a custom screen with drop downs to select research id , group id etc according to the data required by the user to be analyzed. For more details into the type of data stored and options on screen refer to the Hadoop and data stores section of this user guide.

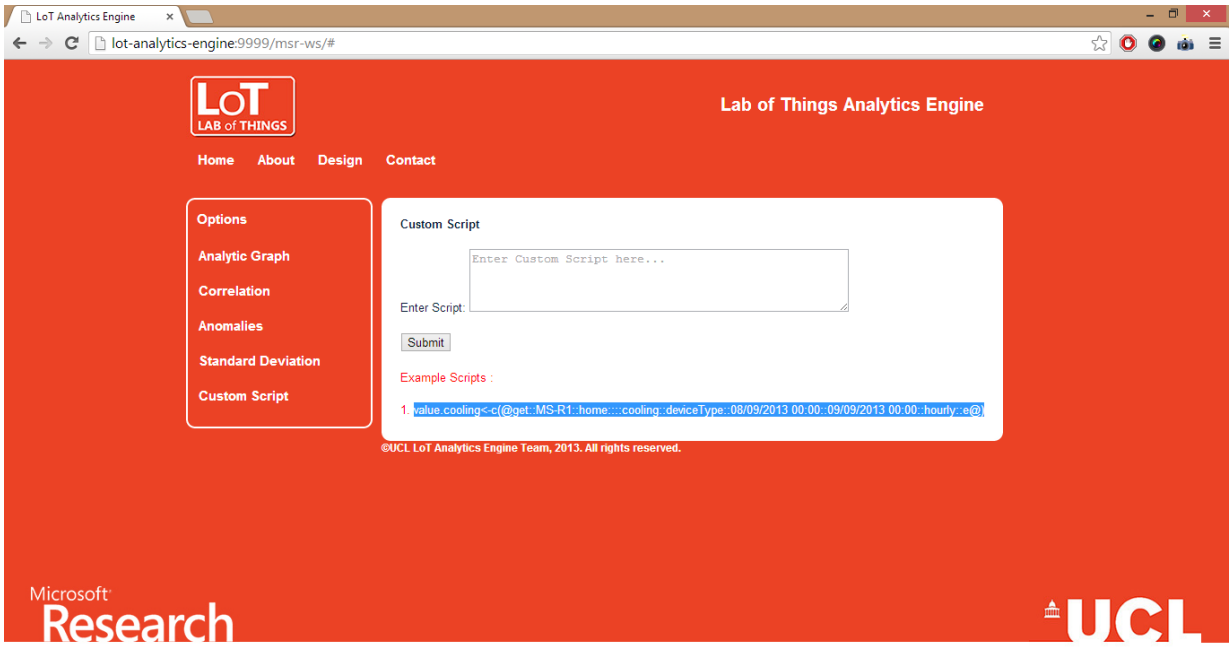
11. To select date and time click on the text box corresponding to that field. A date-time picker will appear. Select the month and year using the arrows and click on the date to select the day. Use the sliders at the bottom of the date-time picker to select the time. The date time picker is shown in the screenshots below.



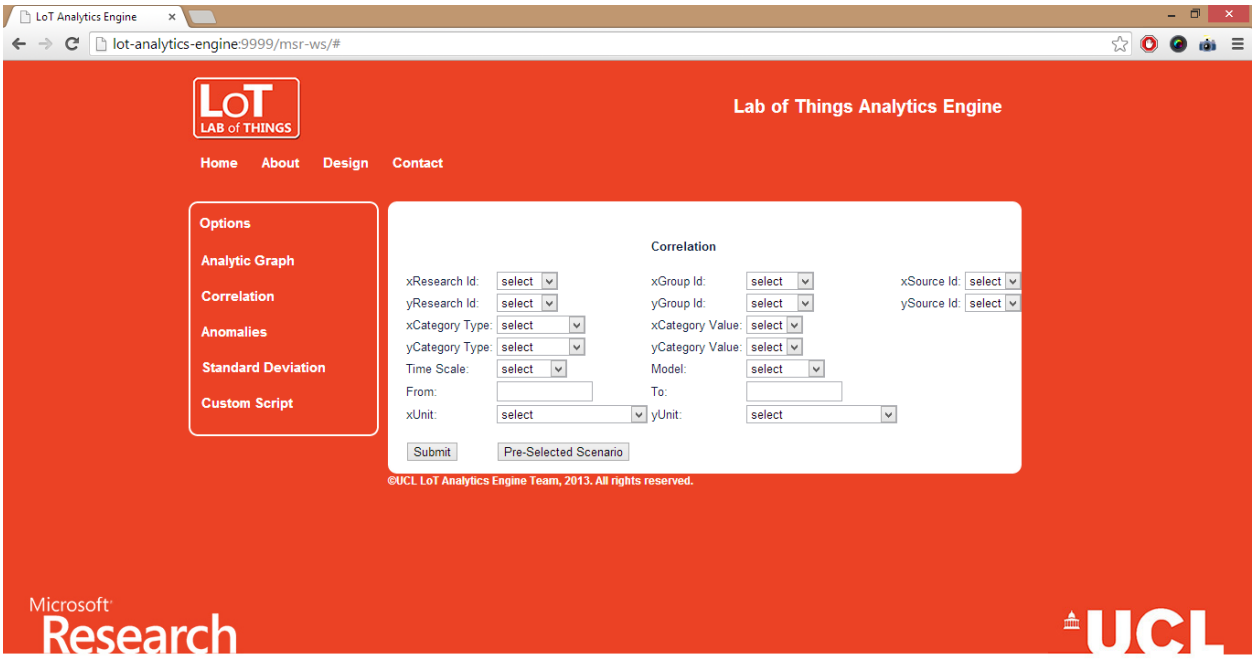
12. After selecting the necessary values and clicking submit the webservices will be sent a request. Depending on the type of graph and statistical model chosen a graph will be displayed on the screen. Below we show an example of the Standard Deviation Model we have provided.



13. The custom script option on the web portal allows the user to enter custom R script to perform analytics on the data through our webservices. We have provided an example R script on the Custom Script Page as shown in the image below.



14. The correlation option is a bit different to the others as it asks users to select two research ids among other options. This is because the correlation function required two data sets to compare.



15. For details on research ids and other options please refer to the data stores section of this user guide.

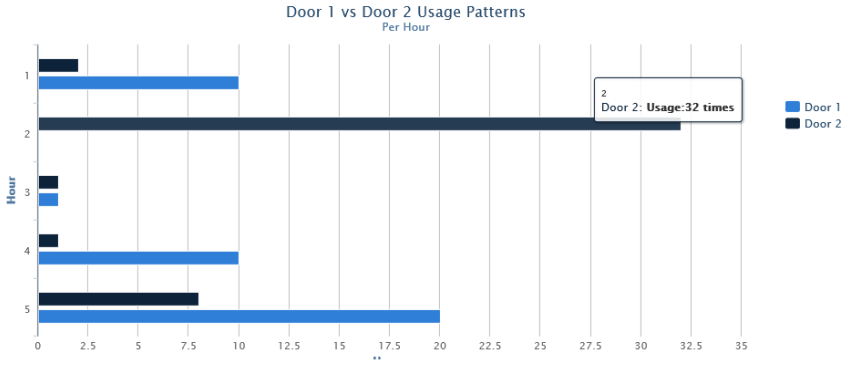
5. Windows 8 Application

1. Download the application source code from the Windows8App Directory.
2. Locate the “HomeOS Analytics.sln” file and open it with Microsoft Visual Studio.
3. For more details about the layout of the app and basic functionality visit http://msdn.microsoft.com/en-us/library/windows/apps/hh758331.aspx and see the guide for the Windows 8 Grid App template. The layout of the application is shown in the image below.

4. For the graphs we are using the Highcharts API, which can be found here http://api.highcharts.com/highcharts.
5. The app contains two scenarios at present one for comparing two door sensors and the other to compare the energy usage of various devices in a users house. The images below show the two scenarios covered so far.

Door Sensor Analytics

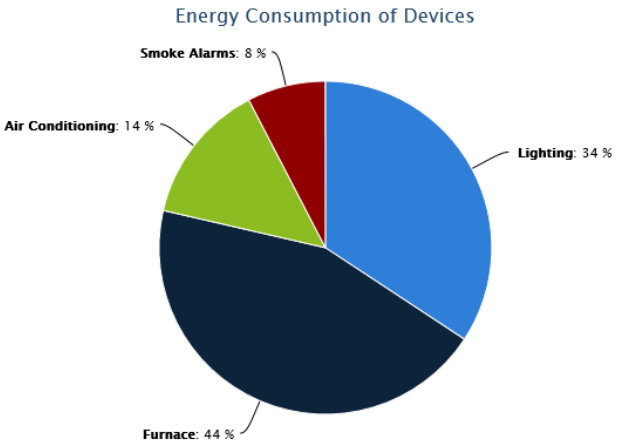
Door Status



The graph displays a comparison of usage patterns of two different doors at the Computer Science Lab in the UCL Malet Place Engineering Building.

Energy Consumption Details

Energy Consumption of Different Devices



The pie chart shows a comparison of energy consumption of various devices in a house.

6. The data source for the app can be configured in the data.js file. Go to the function getData() and change the configuration for the client variable.
7. The door sensor bar chart shown above expects a json in the form {"doorMotionResults":[{"recordTime":"12","countOpen":2,"countClose":10},{"recordTime":"13","countOpen":32,"countClose":0},{"recordTime":"14","countOpen":1,"countClose":1},{"recordTime":"15","countOpen":1,"countClose":10},{"recordTime":"15","countOpen":8,"countClose":20}]}
8. The energy comparison pie chart shown above expects a json in the form : {"deviceComparison":[{"device":"Lighting","consumption":227.03,"timeStamp":"April 1, 2012"},{"device":"Furnace","consumption":34.98,"timeStamp":"April 1, 2012"},{"device":"Air Conditioning","consumption":10.87,"timeStamp":"April 1, 2012"},{"device":"Smoke Alarms","consumption":6.01,"timeStamp":"April 1, 2012"},{"device":"Television","consumption":59.02,"timeStamp":"April 1, 2012"},{"device":"Microsave","consumption":1.03,"timeStamp":"April 1, 2012"}]}

6. HomeOS Publisher Application

1. Create new application named Merge AppMSR1.cs by following “Programming How To” Guide: http://research.microsoft.com/en-us/projects/homeos/programming-howto.docx. and merge created AppMSR1.cs with provided AppMSR1.cs
2. Merge existing DriverZwaveZensys.cs with provided DriverZwaveZensys.cs
3. To add this application into HomeOS platform, modules.xml need to be modified.
4. Connect a Z-wave USB adaptor with your computer, install a driver for this. More details are in a section ‘tip and tricks’ of “Applications and drivers that we developed”, and its link: http://research.microsoft.com/en-us/projects/homeos/whats-included.docx
5. Connect a door sensor device with a Z-wave USB adaptor by setting configuration. The details of configuration for Z-wave devices are explained in “Applications and drivers that we developed” as well.
6. Run HomeOS, and then it will launch MSR1App and the driver for Z-wave
7. Then, HomeOS will collect data from Z-wave device and it will post it into HBase by using REST API. Change the hostname/port of the REST API to where the LoTAnalyticsEngine (or msr-ws.war) is running.

7. Change the Hadoop Cluster

Currently, we point the Hadoop Cluster to UCL's Mapr Cluster. We use HBase (which runs on top of Hadoop Distributed File System-HDFS), as source (input data-store) and sink (output data-store). Please refer to the report on more details on the design.

To configure our system to point to a different cluster (say Azure's HDInsight), one needs to make following changes to the code:

1. **Hadoop project** (could be found on root of project's source code on CodePlex)
   You can change the cluster configuration at msr. MSRDao.java at line 36:

   ```
   conf.set("hbase.zookeeper.quorum", "mapr-m3-01.cs.ucl.ac.uk,mapr-m3-02.cs.ucl.ac.uk,mapr-m3-03.cs.ucl.ac.uk");
   ```

   Also the default name of data-stores are msr-lot-raw and msr-lot-processed, which can be changed at msr1. GenericAvgHbaseDataSrcSink.java:

   ```
   private static final String TABLE_RAW_DATA = "msr-lot-raw";
   private static final String TABLE_PROCESSED_DATA = "msr-lot-processed";
   ```

   You can recreate the jar after making these changes as instructed in Section 2 of this document.

2. **LoTAnalyticsEngine project** (could be found on root of project's source code on codeplex)
   LoTAnalyticsEngine uses the cluster, through HBase queries, to store the raw data and extract the Hadoop processed data.
   Cluster is configured at com.lot.ws.dao.MSRDao.java at line 57:

   ```
   conf.set("hbase.zookeeper.quorum", "mapr-m3-01.cs.ucl.ac.uk,mapr-m3-02.cs.ucl.ac.uk,mapr-m3-03.cs.ucl.ac.uk");
   ```

   You must recreate the msr-ws.war as instructed in Section 1 of this document.

3. Data must be posted to the system (through POST API in from the new war created in step 2) and Hadoop GenericAvgHbaseDataSrcSink job must be run, using the jar created in step 1, on the cluster configured.