Deployment Data Cleanup

D Goldsmith, R Wilkins

December 18, 2012

1 Introduction

Below is a description of the data summary process, how the information is processed an yields calculated.

It takes the form of a walk though of the yield calculation process, and includes code snippets from the R script used to generate the data.

The document was generated in R and Latex, using the Knitr package

2 Summary Table

A new table added to the database, the summary table is intended to hold summary statistics on deployments. This means that future work can avoid having to process entire data sets when dealing with yield, or other summarised functions.

The summary table takes the same form as the reading table, with an additional *summary type* column, these summary types are taken from a lookup table in the database.

Database Rows, and expected inputs are given below

Row	Type	Description
Time	PK,Required	Timestamp of summary, In general I would expect this to use midnight to summarise a complete day. However, if more detailed summaries (such as hourly) are needed, this should not be a problem.
nodeId	PK,Required	Id of node that this summary is from
sensorTypeId	PK	Id of sensor that this summary is from, this can be left NULL to indicate whole node summary samples (for example yield)
summaryTypeId	FK	Id of summary type.
locationId	FK	Id of location this node is from, to keep parity with the reading table
value	float	Value of the summary
textValue	string(30)	Optional text description of the summary, for example “Hot” if we are dealing with exposure graphs.

Table 1: Summary Table Description

3 Scripts

This section has a description of the scripts used process the data, and combine all samples into one database.

These scripts are designed to work with the new format (location aware) database format.

They can found in the *dataclense* directory of the *cogent-house/djgoldsmith-devel* repository.

processCC.py Transfers current cost data from the old style sqlite database, into the new format database.

processAr.py Transfers data from an Archrock postgresql database into the new format database.

getStats.R R script that calculates yields for each deployment in a given database

calcKwh.R R script to caluclate KwH usage from current cost readings.

Further details of these scripts are given below

4 getStats.R

This script calculates summary statistics for all houses in a given database. The statistics are output in two formats.

- *.csv* file with summary output for this database
- update rows in the *summary* table given these statistics

To run the script modify the source file with the relevant database access name. Then run the script through R.

4.1 Script initialisation

- Load the relevant R librarys
- Connects to the database
- Loads the Relevant Lookup tables into memory.
  - Houses Table
  - Sensor Table (For Calibration)
  - Sensor Type Table
  - Summary Type Table

```
# Load Relevant Libraries
library(RMySQL)
library(ggplot2)
library(plyr)

# Setup Database Connection
drv <- dbDriver("MySQL")
# con <- dbConnect(drv,dbname='mainStore',user='chuser')
con <- dbConnect(drv, dbname = "mainStore", user = "root", password = "Ex31S4ga")

# Load the Relevant lookup tables into memeory
allHouses <- dbGetQuery(con, statement = "SELECT * FROM House WHERE address != 'ERROR-DATA'")
summaryData <- dbReadTable(con, "SummaryType")
calibrationData <- dbReadTable(con, "Sensor")
sensorType <- dbReadTable(con, "SensorType")

## Sensors we are interested in (For Yield Calculateions)
sensorTypeList <- subset(sensorType, name == "Temperature" | name ==
    "Humidity" | name == "Light PAR" | name == "Light TSR" | name ==
```

```
    "CO2" | name == "Air Quality" | name == "VOC" | name == "Battery Voltage" |
    name == "Power")

# Create a temporary table to hold summary informtion
houseData <- data.frame(address = allHouses$address, dbStart = NA, dbEnd = NA,
    dataStart = NA, dataEnd = NA, totalNodes = NA, coNodes = NA, yield = NA,
    yieldSD = NA, yieldMin = NA, yieldMax = NA, totalSamples = NA, yieldDays = NA)

# Choose a particular house (for the demo)
i <- 15
THEHOUSE <- allHouses[i, ]
hseName <- THEHOUSE$address
# Output the House for Debugging
print(hseName)

## [1] "Brays lane"
```

At the end of this we have :

1. A connection to the Database
2. A collection of lookup tables used later in the application
3. One main dataframe, to hold the summary information generated during the summarising process.
4. Selected a house, in this case Brays lane

4.2 Loading the Inital Dataset

This section of code fetches the relevant data for each property.

First the relevant house is fetched, and the start and end dates processed to convert to a format R is comfortable with.

Next the Relevant locations are fetched using the following query This query also fetches the corresponding room names for each location for use in a lookup table if required.

```
# As the function parameter is a row number, Work out which House
# we are dealing with. print(paste('Processing House ',hseName))
rowNo <- which(houseData$address == hseName)
cleanName <- gsub(" ", "", hseName)

# Get House from the database
houseQry <- paste("SELECT * FROM House WHERE address = '", hseName, "'",
    sep = "")
theHouse <- dbGetQuery(con, statement = houseQry)

# Process start and end dates into a format that R is happy with
theHouse$sd <- tryCatch({
    as.POSIXlt(theHouse$startDate, tz = "GMT")
}, error = function(e) {
    NA
})

theHouse$ed <- tryCatch({
    as.POSIXlt(theHouse$endDate, tz = "GMT")
}, error = function(e) {
    NA
})
```

```
# Build the Location Query
locQry <- paste("SELECT * FROM Location as Loc ", " LEFT OUTER JOIN Room as Room ",
    " ON Loc.roomId = Room.id ", " WHERE houseId =", theHouse$id, " AND Room.name NOT LIKE 'PhyNet%'
    " AND Room.name NOT IN ('Hot Water','Cold Water','Flow','Return','Hot','Cold','HotWater','ColdWa
    sep = "")

# Fetch Locations from the DB
locations <- dbGetQuery(con, statement = locQry)

# Convert into a string so we can put the data into the next query
locationIds <- paste(locations$id, collapse = ",")
```

Next the relevant dataset is fetched. Rather than fetch the entire dataset, and process it on the local machine, the query used makes use of SQL summarising functions to group the data by day. The query fetches the following values:

nodeId Used to Identify the Node

locationId Used to Identify the Location

date Timestamp converted to a date (i.e. with the time removed)

minTime Time of first sample on this date

maxTime Time of last sample on this date

minVal Minimum reported Value on this date

maxVal Maximum reported value on this date

meanVal Average value reported on this date

```
houseData[rowNo, ]$dbStart <- theHouse$startDate
houseData[rowNo, ]$dbEnd <- theHouse$endDate

# Setup the Query
sTypes <- paste(sensorTypeList$id, collapse = ",")
dataQry <- paste("SELECT nodeId, type, locationId, DATE(time) as date,count(*) as count,",
    " min(time) as minTime,max(time) as maxTime,", " min(value) as minVal, max(value) as maxVal, avg
    " FROM Reading WHERE locationId IN (", locationIds, ") ", " AND type IN (",
    sTypes, ") ", " GROUP BY nodeId,type,DATE(time)", sep = "")

# Fetch the data
houseSummary <- dbGetQuery(con, statement = dataQry)
# Add a time object so that makes sense to R
houseSummary$dt <- as.POSIXlt(houseSummary$date, tz = "GMT")
houseSummary$DT <- as.Date(houseSummary$dt)
# And add some extra columns to hold summary information on bad
# samples
houseSummary$countWithBad <- houseSummary$count
```

We should end up with a load of uncalibrated data as shown in figure **??**

```
# Plot the Uncalibrated Data
plt <- ggplot(subset(houseSummary, type == 0))
plt <- plt + geom_point(aes(dt, meanVal))
plt <- plt + geom_errorbar(aes(dt, ymin = minVal, ymax = maxVal))
```

```
plt <- plt + opts(title = paste("Uncalibrated Temperature data for ",
    hseName))
plt <- plt + xlab("Date") + ylab("Reading")
plt + facet_grid(nodeId ~ .)
```

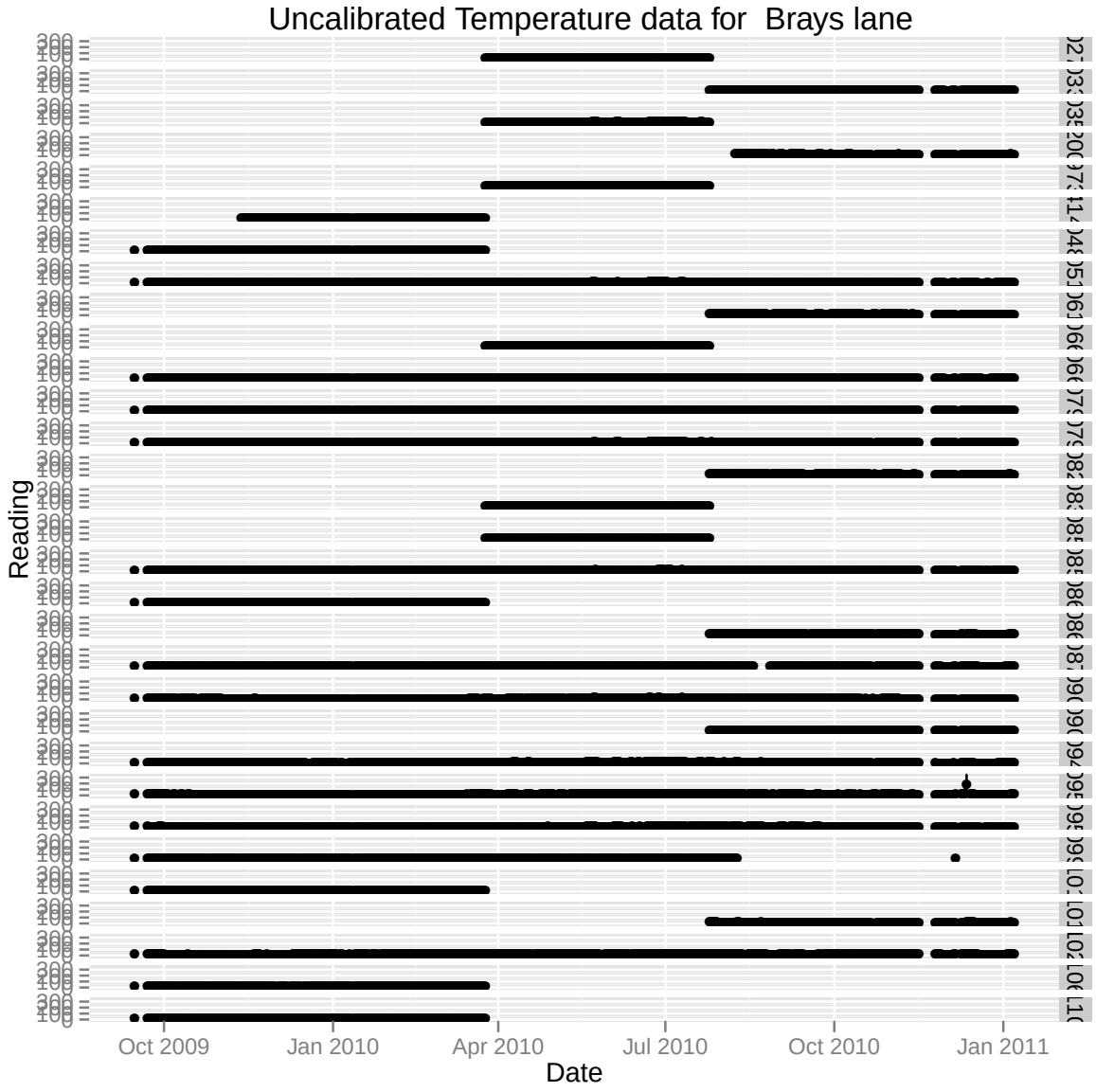

Figure 1: Uncalibrated Data