

# DataYours



@akbooper, February 2016

DataYours	1
Introduction	2
Quick Start	3
Configuration	4
Basic operation	5
Plotting Data	6
Selecting data for recording	7
Storing Graphs	9
Converting from previous DataYours7 version	10
Advanced configuration	10
Multiple processors	11
Integration with dataMine	11
Handling non-numeric data	12
DataYours as an AltUI Data Storage Provider	13
Design notes:	15
Multi-resolution: multiple archives in one file	16
The storage schema and aggregations rules	17
The re-write rules	17
The Daemons	18
DataWatcher	19
DataCache	21
DataGraph	22
DataDash	23
DataMineServer	23
Acknowledgements	25
References	25

## Introduction

DataYours is a plugin to acquire, store, and display data about Vera devices and measurements. The plugin itself is really just a framework, library, and launcher for other modules ('daemons') which implement the real functionality.

Four modules (DataWatcher, DataCache, DataGraph, DataDash) provide an implementation of the open-source Graphite system for storing and plotting time-based data. According to the official documentation: *"Graphite is an enterprise-scale monitoring tool that runs well on cheap hardware."* This has been re-engineered in pure Lua to run on the MiOS / Vera system, and packaged as a DataYours. Additionally, a DataMineServer module brings a graphical interface to dataMine channels and graphs within the DataYours environment.

The real-time aspects of the system (data capture and storage) are handled by the data 'daemons' which take up very little space and cpu resources. The Graphite system database is called Whisper and is a 'round-robin' structure: that is, it only stores data for a finite time (and on uniform increments of sampling time - the finest resolution being one second) and never grows its disk space usage (ie. all storage is pre-allocated.) A key feature of Whisper is that it supports multiple archives with different retentions (maximum duration) and resolutions. This effectively enables data compression of what might otherwise be very large database files.

The architecture supports data acquisition and storage over multiple Veras (which need not be 'bridged') and uses a very low-overhead communication protocol (UDP) which is also able to communicate with external databases (such as syslog.) Utilising the CIFS system (separately installed) the Whisper database may be hosted on an external NAS. A 'dashboard' web server provides a user interface and presentation of graphics, but this is quite separate from the underlying system and could easily be replaced by some other implementation.

The database only stores numerical data, although the front-end data capture and syslog forwarding also works for any string data type, and a conversion lookup table can be provided on a per-variable basis to convert to numeric values. No special plotting is provided for energy usage, although a DataWatcher option is to use the built-in 'live\_evergy\_usage' Luup functionality to report individual power usage on a periodic basis.

For easy integration with dataMine, the dashboard provides read-only access to all the dataMine database including stored graphs, and has the capability to plot data from both the dataMine and Whisper databases.

For easy integration with AltUI, the DataWatcher daemon registers itself as a local Data Storage Provider, enabling data to be recoded and logged through the AltUI interface.

## Quick Start

If you don't want to bother with the following documentation, then try this:

1. Install `DataYours` from the App store.
2. change the `DAEMONS` variable of `DataYours` to `'Watch Cache Graph Dash'`, also add `'Mine'` if you want to interface to `DataMine`.
3. configure the Whisper database location by setting `LOCAL_DATA_DIR` variable to point to your desired Whisper database directory (this parameter should start and end with a `/`, eg. `/nas/whisper/`)
4. configure `Mine` (if you have it) by setting its `DATAMINE_DIR` variable to your desired `dataMine` database directory (usually `"/dataMine/"`).
5. If you have them, copy the files `storage-schemas.conf` and `storage-aggregation.conf`, into your target Whisper database directory.
6. Start investigating menu buttons on the dashboard page (tool tips for most items):  
`http://[VeraIP]:3480/data_request?id=lr_dashboard`
7. `Devices` should show you all the devices on your system, drilling down using the view device variables tool tip link will show you all that device's variables. Tool tips there show current values. Perhaps read the section below on how to navigate TreeMaps.
8. `whisper` will initially be blank, but once you start recording variables to the database, they show up here. Tool tips then show various plotting options.
9. `Graphs`, initially also blank, shows saved plot configurations which can have multiple variable plots and be displayed over different time periods.
10. `Graphs` are configured from the `select` tool tip menu on items on the Whisper page.
11. If the `dataMine` modules are installed and the location of the `dataMine` database set in the configuration file, then a pair of `dataMine`-related menu buttons appear and give access to all the stored variables and graphs from there.

Note that Whisper stored graphs show ONLY Whisper variables, `dataMine` plots and stored graphs show ONLY `dataMine` variables.

## Configuration

Configuration of the system is done almost entirely through a few key device variables of DataYours. User-configurable parameters are all UPPER CASE. Others should not be changed.

CONFIG_DIR	location of some .conf files (/www/)
DAEMONS	daemons to launch (Watch Cache Graph Dash Mine)
DATAMINE_DIR	dataMine database location (/dataMine/)
DESTINATIONS	IP and Port of Cache(s) (127.0.0.1:2003)
ICON_PATH	icon URL (UI5: /cmh/skins/default/icons/) (UI7: /cmh/skins/default/img/devices/device_states/)
LIVE_ENERGY_USAGE	set to '1' to report power usage of devices
LOCAL_DATA_DIR	Whisper database location (eg. /nas/whisper/)
MEMORY_STATS	set to '1' to report processor system memory usage
SYSLOG	syslog IP and port for DataWatcher (eg.172.16.42.112:514)
UDP_RECEIVER_PORT	UDP port for Cache to listen on (2003)
VERAS	list of IPs or remote Veras (eg. 172.16.42.10:3480, ...)

These device parameters override the conventional configure file `carbon.conf` used by the original system. It is possible to launch the individual daemons without running the DataYours plugin, for a really minimal system footprint, in which case the configuration parameters are read from `carbon.conf`.

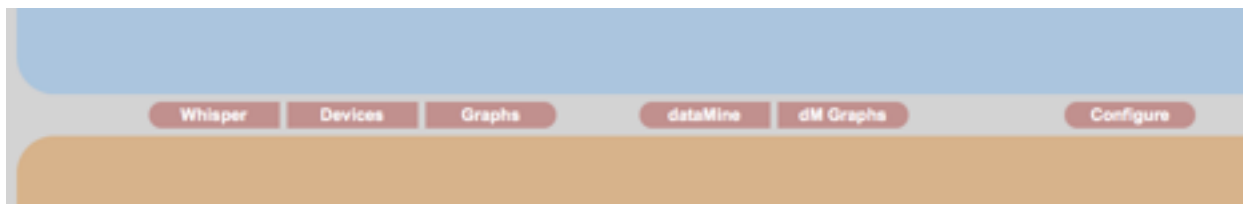
## Basic operation

The DataDash daemon provides an integrated user interface.

`http://<VeraIP>:3480/data_request?id=lr_dashboard`

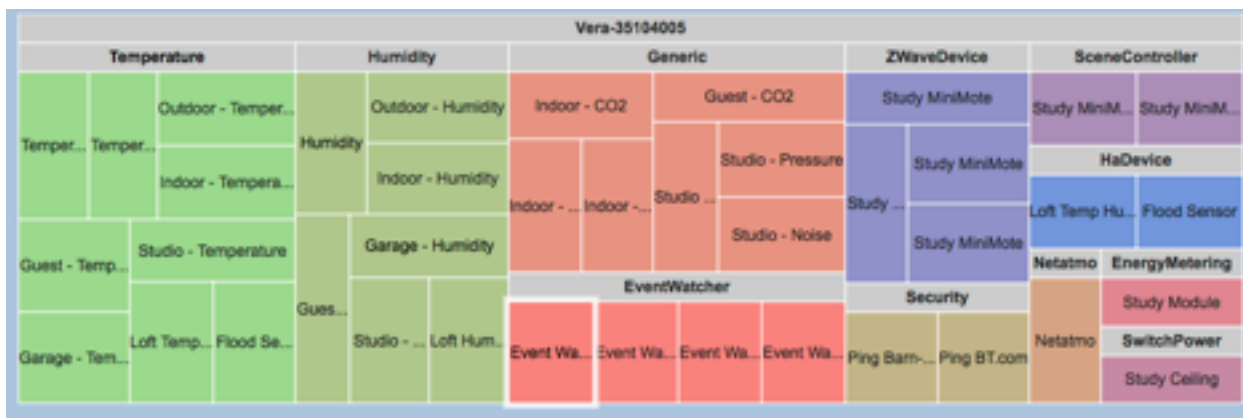
On first arrival here, the screen is almost blank, with just a few buttons across the centre.

(If dataMine is not configured, then that button group does not appear.)



The top half of the display is reserved for 'TreeMaps'. These are space-filling rectangular panels which can represent groups of watched variables, devices, stored graphs, and the like.

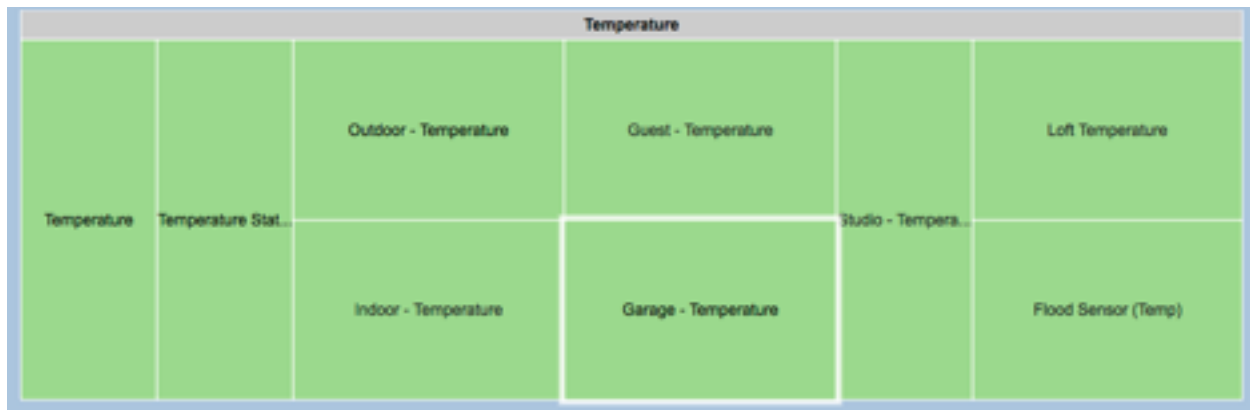
The basic operation of all the different TreeMaps is the same. The data they show is actually a tree structure (hence the name).



In this example (a well-populated Whisper database with 40 stored variables), shows all the variables being logged into Whisper on one screen, grouped by Vera (in this case, only one, the local machine) and then by deviceId (and colour-coded) along with their device - variable names.

To zoom in to a particular group of variables, click on the area of interest (either directly on a measurement or on its grey category title) - Temperature, for example:

To zoom out to a higher level, right-click (you may need to do this several times to reach the top level again.) It's not necessary to zoom, but it can be helpful.



Mouse over on one of the rectangles and a tool-tip menu pops up with the name of the variable, some other information, and options to plot it over the last day / week / month.

**[324] Garage - Temperature**  
 10m:7d,1h:30d,3h:1y,1d:10y [0.0] average  
 plot: [day](#) / [week](#) / [month](#) / [select](#)  
 Temperature - CurrentTemperature

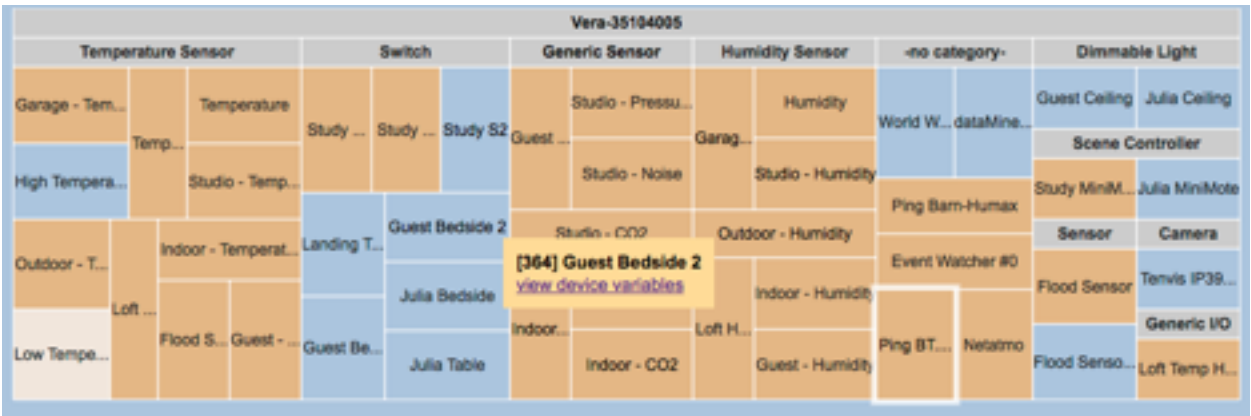
The bold number in square brackets is the device id. The second line shows all the archives configured for storing this particular variable's history. In this case, 10 minute sampling for one week (7 days), one hour sampling for one month (30 days), three hour samples for one year, and finally daily sampling for 10 years. An averaging function is used to aggregate data between the archives, and the [0.0] value is the 'xFilesFactor' described elsewhere in the Whisper documentation.

## Plotting Data

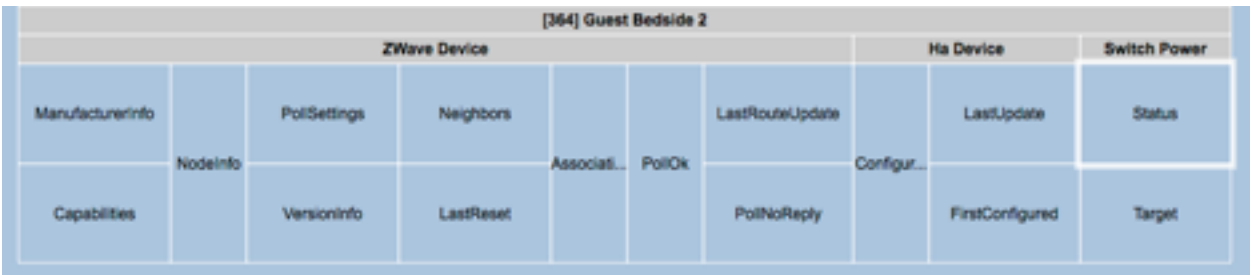
The day / week / month links on the above tool tip menu do exactly what they say for the current variable. 'select' brings up a menu page which allows you to accumulate a number of selected variables into a selection list and then store them, with various plotting options, for later use. See later section on this.

# Selecting data for recording

The Devices TreeMap, similar in operation to the above, shows all the devices on the known Vera systems (just one shown in this case.)



Here, the tool-tip gives only one option: `view_device_variables`. This brings up a TreeMap for all the variables of a single device, showing, in another tool-tip, the current values as you mouse over each one.



Hence with **just one click** from the Devices TreeMap, the value of **any variable** on any configured system can be viewed.



Having located a variable of interest, the `watch` option, creates a Whisper database set of archive according to the stored rule-base and then displays a large button to press to initiate watching that variable.

In the event that the variable/serviceId selected does not have a matching rule in the rule-base, selection of these parameters has to be done manually.

To make it simple, there are menus for archive durations covering periods from one day to ten years, with a variety of sample rate options on each of them. Just one, or many, archives may be configured for a single variable.

The screenshot shows a web interface for configuring a Whisper database. On the left, there are two buttons: 'CREATE' (in a red circle) and 'Reset' (in a grey circle). The main area is titled 'Storage Aggregation' and contains a row of buttons for the aggregation method: 'average' (highlighted in red), 'sum', 'last', 'max', and 'min'. Below this, the text 'Whisper Database: MIOS\_35104860.088.um:akbooe-com:serviceId:Netatmo1.CalibrationOffset' is displayed. The 'Storage Archives' section contains a table with six rows representing different archive durations: 'day', 'week', 'month', 'quarter', 'year', and 'decade'. Each row has a red button with a minus sign, followed by several input fields for configuring the archive's resolution and retention. For example, the 'day' row has fields for '1s:1m,1m' and '1m'. The 'decade' row has fields for '1d:5y' and '1d:10y'.

The storage aggregation part of the form allows selection of the aggregation method to choose to apply between the archives - **it is VERY important to pick at least one archive** - read on!

The underlying Whisper database has an essentially infinite number of options to use in configuring archives (there are also an infinite number of illegal configurations because there are strict rules about the duration and sampling rates between adjacent archives.) For full details, see <http://graphite.readthedocs.org/en/latest/whisper.html>, but to summarise the information there:

*“Whisper databases contain one or more archives, each with a specific data resolution and retention (defined in number of points or max timestamp age). Archives are ordered from the highest-resolution and shortest retention archive to the lowest-resolution and longest retention period archive.”*

*“The total retention time of the database is determined by the archive with the highest retention as the time period covered by each archive is overlapping. That is, a pair of archives with retentions of 1 month and 1 year will not provide 13 months of data storage as may be guessed. Instead, it will provide 1 year of storage - the length of its longest archive.”*

*“Whisper databases with more than a single archive need a strategy to collapse multiple data points for when the data rolls up a lower precision archive. By default, an average function is used. Available aggregation methods are: average /sum / last / max / min.”*

*“When Whisper writes to a database with multiple archives, the incoming data point is written to all archives at once. The data point will be written to the highest resolution archive as-is, and will be aggregated by the configured aggregation method and placed into each of the higher-retention archives.”*



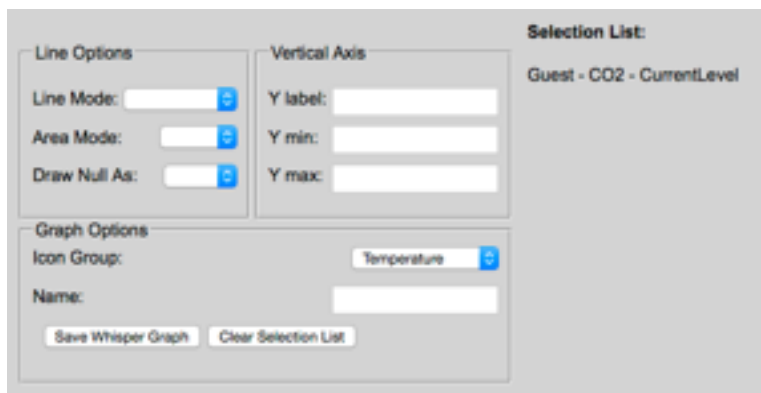
*“When data is retrieved (scoped by a time range), the first archive which can satisfy the entire time period is used. If the time period overlaps an archive boundary, the lower-resolution archive will be used. This allows for a simpler behaviour while retrieving data as the data’s resolution is consistent through an entire returned series.”*

Aside from hiding the complexity of some of this, the Storage Archives menu configures options for archives with typically between 1,000 and 2,000 points in each. This makes retrieval and plotting quite fast (unlike dataMine where it was easily possible to crash the system by asking to plot too much data.)

On a VeraLite, the writing performance approaches 1000 separate data points per archive per second – way more than necessary to keep up with variable state changes on even the largest systems.

## Storing Graphs

The `select` tool tip menu item from the Whisper page brings up the following menu:



Multiple items can be added to the selection list from their own `select` tool tip menus.

This is where graphs are configured for future reuse, just as in dataMine:

- **Icon Group** - mirrors exactly the icon option of dataMine and is simply used to classify plots into groups on the Graph menu (no icons here, actually)

- Plotting modes: Line / Area / Draw Null as Zero perform as described in the Graphite documentation, but are fairly self explanatory. The DataGraph daemon attempts to select the correct plotting mode depending on the aggregation method used for multiple archives. Sometimes it gets it right, sometimes not. These values will override the automatic selection.
- Name the graph before clicking Save
- **Clear empties the selection list. Saving a named graph with nothing in the selection list deletes an existing graph of the same name.**

## Converting from previous DataYours7 version

The device files are named differently from the previous 'Version 7'. You cannot run both systems simultaneously. Whilst a multi-processor system can safely run different versions, and data will be correctly sent and received by DataWatcher and DataCache, the DataDash daemons will not talk to the old version. You are recommended to upgrade all your systems for this reason.

## Advanced configuration

The "Configuration" button on the dashboard page opens a separate browser window:



In the top part are green boxes with icons for each configured Vera and their daemons. Click on any of these (DataWatcher, DataCache, DataGraph, DataDash) to get a listing in the lower part of the display (scrollable) of the configuration for that daemon, including performance statistics and diagnostic messages.

You may be asked to provide this information when diagnosing a problem.

## Multiple processors

To link multiple `DataYours` systems on separate `Veras`, simply add their IP addresses to the `VERAS` device variable under the `Dash` child device.

```
VERAS = 172.16.42.10:3480, 172.16.42.11:3480, 172.16.42.12:3480
```

Additionally, multiple locations can be added to the `Watch` daemons on any of the machines to send updates to multiple `Cache` daemons.

```
DESTINATIONS = 127.0.0.1:2003, 10.0.0.3:2003
```

...always ensuring that the port number (2003) matches that on the destination `Cache` daemon

```
UDP_RECEIVER_PORT = 2003
```

## Integration with dataMine

See the configuration section of this manual for installing the optional `DataMineServer` daemon. New buttons on the dashboard enable plotting of `dataMine` channels and stored graphs.

## Handling non-numeric data

Whilst the back-end Whisper database only handles numeric values, the front-end capture by DataWatcher supports any textual data. It's possible to covert between know text variable values and numeric values by providing a look-up table.

You need to list your symbolic variables by serviceId and variable name in the Data-Translation.conf file which should be located wherever the CONFIG\_DIR point to (by default, /www/, although this is not a great choice.) like this (lines prefixed with '#' are comments):

```
#
# DataYours symbolic data lookup tables
# 2015.03.19 @akboorer
#
# one section for each serviceId.variable lookup
# device number must be wildcard '*'
#
# Example: Thermostat ModeState (Idle, Heating or Cooling)
# [*urn:micasaverde-com:serviceId:HVAC_OperatingState1.ModeState]
# Cooling = -1
# Idle = 0
# Heating = 1
# note other allowed values of:
#   FanOnly, PendingHeat, PendingCool, Vent
# are undefined
#

[*urn:micasaverde-com:serviceId:HVAC_OperatingState1.ModeState]
Cooling = -1
Idle = 0
Heating = 1
#
```

## DataYours as an AltUI Data Storage Provider

DataYours provides a local, or remote, data storage provider within the AltUI framework, and works on Vera and under openLuup on any machine. It enables the use of the AltUI interface to archive and plot device variable history.

DataWatcher will register itself with AltUI, if installed, and be available as a Data Storage Provider under the graphing menu for each variable. A simple naming convention with a one-letter suffix will, by default, configure a data archive for that variable to be:

- .d - one minute resolution for one day
- .w - five minute resolution for one week
- .m - twenty minute resolution for one month (30 days)
- .q - one hour resolution for one quarter (90 days)
- .y - six hour resolution for one year

The database can be configured to be local to Vera (or openLuup) or on a remote machine, through the setting of the LOCAL\_DATA\_DIR parameter.

The mapping from name extension, like .m, to the archive structure is done with the standard DataYours configuration file `storage_schemas.conf` which should be located in LOCAL\_DATA\_DIR. In the following code snippet are the schema rules which will implement the above mapping. They can, of course, be changed to suit. These rules are only used in the creation of new files, so it is also possible to create a specific archive structure for a particular metric and just use that name without any extension modifier.

```
#
# Schema definitions for Whisper files. Entries are scanned in order,
# and first match wins. This file is read whenever a file create is required.
#
# [name] (used in log reporting)
# pattern = regex
# retentions = timePerPoint:timeToStore, timePerPoint:timeToStore, ...

# 2016.01.24 @akboer
# basic patterns for AltUI Data Storage Provider

[day]
pattern = \.d$
retentions = 1m:1d
```

```
[week]
pattern = \.w$
retentions = 5m:7d

[month]
pattern = \.m$
retentions = 20m:30d

[quarter]
pattern = \.q$
retentions = 1h:90d

[year]
pattern = \.y$
retentions = 6h:1y
```

Since DataYours is inherently a modular, distributed system, the data may be replicated on local and remote databases. It is also possible to run without DataYours installed on the local system at all, sending everything to a remote machine, but there needs to be a one-time manual registration of the remote machine as an AltUI Data Storage Provider. The AltUI interface for saving and plotting the data remains, in either case, the same (and is far more straight-forward than the dashboard interface provided by DataYours.)

## Design notes:

This prototype design is based heavily on the Graphite system (itself written in Python). There is some great documentation for this here <https://graphite.readthedocs.org/en/latest/overview.html>, and much of the high-level information is valid for DataYours. The database itself is a round-robin style, called Whisper <https://graphite.readthedocs.org/en/latest/whisper.html> with fixed maximum duration, but possibly varying resolution as data ages (see the docs!)

The Whisper code, originally written in Python, has been translated to Lua, and then refactored somewhat. The database code is pure Lua and will run anywhere, but it is not binary-compatible with Graphite Whisper files because I have chosen CSV rather than binary packing. This makes them exactly three times larger than real Whisper ones, but space (outside of Vera) is not a problem.

The Graphite daemons, which provide all of the original system's functionality have not been ported because they have very heavy external dependencies and a complex install process - quite unsuited to the Vera environment. Their basic functionality has been reverse-engineered whilst endeavouring to maintain their external interfaces. It should be quite possible, therefore, to interface this code with a real Graphite system and use all its plotting utilities (and third-party ones.)

The whole thing is very modular. There are three major components (daemons) at present:

- **DataWatcher** - simply watches for variable changes and sends them on (essentially a 'Carbon-relay' with a Vera/Z-Wave bridge)
- **DataCache** - a fairly complete implementation of Graphite's 'Carbon Cache' which receives and stores data in Whisper
- **DataGraph** - a partial implementation of Graphite's 'Web App' which plots results from the database

All data passes between components in UDP datagrams in the standard **Whisper plaintext format**, so in fact all the components can be running on different machines. I have three machines with three DataWatchers each feeding two DataCache instances on different machines and DataGraph plotting from another machine accessing the data stored on a NAS. All the daemons also listen for HTTP requests.

Both data and diagnostic messages can be written to an external syslog server instead of Vera's log file. If you just wanted to send data to syslog, then the only component you need is DataWatcher.

## Multi-resolution: multiple archives in one file

It's really important to appreciate that you can't simply scale file size requirements linearly if you are using this multi-archive feature, because older data are stored with lower time precision. As an example, on my system I am storing power usage in a file which has a storage schema of "20m:30d, 3h:1y, 1d:10y", or:

- 20 minute, for 30 days
- 3 hourly, for 1 year
- once a day for 10 years

Stored at full 20 minute sampling, this would be  $3 \times 24 \times 365 \times 10 = 262,800$  points, at 36 bytes a sample, that's ~10 Mbytes (OK, not too big), but in this multi-scale archive it only takes about 300Kb - that's 30 times less! You have to think carefully about how you will use the data in future before configuring an archive.

The more complex multi-archive structure I've chosen for my security sensors serves to demonstrate the sophistication possible with Whisper file archives:

"1s:1m, 1m:1d, 5m:7d, 1h:90d, 6h:1y, 1d:5y" with "sum" aggregation

*Explanation:*

Security sensors, specifically PIRs, generate data on a frequent basis. If two activations (transitions to "1") occur within the resolution of one sample as configured in the database for that sensor, the first one will not be seen – the second simply overwrites the first (they would both have the same timestamp anyway.)

So the sampling of the first archive is set to one second, the finest possible in Whisper. Not really caring about information on this small timescale, the next archive switches to one minute sampling for a whole day. Then five minutes for a week, then an hour for three months...

The aggregation function used between the archives is 'sum' so that if, for example, three activations occur in one minute, then the second archive records '3' for that minute, after a day, the minute totals get summed into 5 minute bins, and so on.

The round-robin nature of the archives and their multi-resolution capability bring some fantastic possibilities. But care is needed to understand what the aggregation function is doing to the data.



## The storage schema and aggregations rules

Read the Graphite documentation on these files.

<http://graphite.readthedocs.org/en/latest/config-carbon.html#storage-schemas-conf>

<http://graphite.readthedocs.org/en/latest/config-carbon.html#storage-aggregation-conf>

The implementation is complete.

In particular, note that these are (a subset of) regular expressions used to match incoming data series names.

The rules are scanned in order, with the first matching one being used.

If a match is found, then the Whisper archives are created automatically, according to those rules.

If no match is found then the archive configuration menu is presented to the user for manual selection of archives.

One important departure from the official Graphite documentation is the location of the files `storage-schemas.conf` and `storage-aggregation.conf`: they should be placed into your target Whisper database directory.

## The re-write rules

The DataCache daemon implements the re-write rules functionality of the carbon aggregator daemon (but not the aggregation functions)

Read the Graphite documentation on this file (which, as for the above .conf files should be places in the Whisper database directory at LOCAL\_DATA\_DIR.)

<http://graphite.readthedocs.org/en/latest/config-carbon.html#rewrite-rules-conf>

## The Daemons

All of the components run as autonomous (and asynchronous) daemons - not taking their own additional stack space or presenting any device features. They are small and, hopefully, efficient. A single framework module **DataDaemon** (~400 Lua lines) provides all its clients (DataWatcher, DataCache, etc.) with some basic features:

1. HTML URL command line interface
2. persistence of configuration parameters across Luup restarts and Vera reboots
3. individual configuration commands and files for each client
4. a listener callback for incoming UDP datagrams (containing pathname, value, and timestamps)
5. a UDP sender call for data to be sent to one or many remote listeners (so it can replicate incoming data)
6. optional logging of data and debug information to a remote syslog server

The URL configuration interface provides a number of commands to support this functionality. All configuration commands are sent to the daemon as URL GET requests of the following form:

```
http://<yourVeraIP>:3480/data_request?id=lr_<Name>&<a=b>
```

where Name is the name of the client and <a=b> is client dependent, but all clients support **show=config** - listing an internal configuration page (good for diagnostics)

Client-specific configuration parameters are kept in a client-specific .conf file which is loaded at startup. These files are named like DataWatcher.conf etc., stored in /www/, or wherever the DataYours parameter CONFIG\_DIR points to, so are available to any web browser from the normal Vera IP address.

This basic (and light-weight) framework provides all the I/O and persistent context that the various client modules need.



## DataWatcher

DataWatcher is simply the data collection front-end, but has a counterpart in the Graphite system as the `Carbon-relay` daemon. Effectively, DataWatcher relays watched variable changes to any number of listening DataCache daemons on local or remote machines using a UDP protocol. It's also able to send data to a syslog server, watch power usage of attached devices, and report on system memory usage.

A couple of URL command requests allow configuration of which variables to watch.

Device variables are uniquely specified through the syntax `<deviceNo>.<serviceId>.<variable>` on the command line:

```
http://<yourVeraIP>:3480/data_request?
id=lr_DataWatcher&watch=321.urn:upnp-org:serviceId:Temperature-
Sensor1.CurrentTemperature
```

which starts watching that variable.

Wildcard (\*) device numbers may be used to specify all devices with that serviceId and variable name:

```
http://<yourVeraIP>:3480/data_request?
id=lr_DataWatcher&watch=*.urn:upnp-org:serviceId:Temperature-
Sensor1.CurrentTemperature
```

A second command `nowatch` stops logging of a specific variable:

```
http://<yourVeraIP>:3480/data_request?
id=lr_DataWatcher&nowatch=321.urn:upnp-org:serviceId:Temperature-
Sensor1.CurrentTemperature
```

This is useful to deselect unwanted devices picked up when using wildcard device numbers, particularly if being used interactively.

The UDP format is very lightweight, uses minimal network and cpu resources, but doesn't guarantee receipt of the message since there is no handshake. However, it's very widely used for non-critical data and it as reliable as the underlying network (which, on an internal LAN in particular, should be very reliable.)

The DataWatcher daemon is quite simple, taking ~300 lines of Lua code to add its specific functionality to the generic daemon code.

**If all you want to do is to send variable changes to syslog or some other UDP server then DataWatcher is all you need.**

## SYMBOLIC VALUE HANDLING

In order to convert device variables with symbolic values into numbers which can be stored in the Whisper database, you need to list your symbolic variables by serviceId and variable name in a configuration file `DataTranslation.conf` (which should be located in the directory pointed to by the `CONFIG_DIR` variable, this defaults to `/www/`)

(lines prefixed with '#' are comments)

```
#
# DataYours symbolic data lookup tables
# 2015.03.19 @akboer
#
# one section for each serviceId.variable lookup
# device number must be wildcard '*'
#
# Example: Thermostat ModeState (Idle, Heating or Cooling)
# [*.urn:micasaverde-com:serviceId:HVAC_OperatingState1.ModeState]
# Cooling = -1
# Idle = 0
# Heating = 1
# note other allowed values of: FanOnly, PendingHeat, PendingCool, Vent
# are undefined
#

[*.urn:micasaverde-com:serviceId:HVAC_OperatingState1.ModeState]
Cooling = -1
Idle = 0
Heating = 1
#
```



## DataCache

DataCache is a look-alike for Graphite's Carbon-cache daemon. It simply listens for incoming data and stores it in the Whisper database.

Actually, it's not quite that simple, because there could be some complex configuration decisions to be specified (see the Graphite Carbon documentation <https://graphite.readthedocs.org/en/latest/config-carbon.html>) DataCache is a fairly complete implementation of Carbon-cache, although the Carbon-aggregator functionality is limited to applying re-write rules (ie. no actual data aggregation.)

The incoming datagram specifies all the metadata that is needed to know where to store it: the `<deviceNo>.<serviceId>.<variable>` syntax of the data source is simply mapped to a filename. (I've chosen not to replicate the directory tree which the original carbon-cache uses. It's designed for tens of thousands of variables, but we'll only be storing a few hundred at maximum, so DataCache stores all the Whisper files in a single directory, making administration and backup really very easy.)

But what about new data? Whisper files have different time resolutions and data retentions, possibly containing a number of archives with progressively longer retention times and lower resolutions (see the docs: <http://graphite.readthedocs.org/en/latest/whisper.html>) and aggregation functions to apply when down-sampling. These have to be specified initially for a new file to be created.

In Carbon-cache there are a number of different configuration rule sets which define these: the configuration files `storage-schemas.conf`, `storage-aggregation.conf` and `rewrite-rules.conf` work exactly as described in the above documentation. They should be placed in the same directory as the rest of the Whisper database. They are optional in the DataYours configuration since this forces manual specification of database archives prior to storing any data.

DataCache configuration is defined in the `[cache]` section of the `carbon.conf` file, although normally overridden by the DataYours device configuration parameters.

This is implemented in ~200 lines of Lua code (separately from the generic Whisper library.)



## DataGraph

DataGraph is the Graphite Web App look-alike, see:

[http://graphite.readthedocs.org/en/latest/render\\_api.html](http://graphite.readthedocs.org/en/latest/render_api.html)

It reads data from the Whisper database and plots or prints it out in a variety of formats.

```
http://<yourVeraIP>:3480/data_request?
id=lr_render2&target=Vera-12345678.321.urn:upnp-
org:serviceId:TemperatureSensor1.CurrentTemperature
```

This is currently only a partial implementation, so following along with the Graphite render API documentation link, the exceptions are here below:

- **target** -
  - single paths and value lists {...} supported,
  - no wildcards (\*) or character lists [...]
- **from/until** - relative time (the most useful) works as described. For absolute time formats, Graphite did not, regrettably, use the ISO 8601 standard, which is what is currently implemented in DataGraph. However, the good news is that the YYYYMMDD format is the same in both, so stick to that.
- **format** - supports svg (the default), csv, and json, in exactly the formats described, except that the svg option does not embed the metadata variable (makes no difference if you're just looking at the plot.) raw, png, and pickle are not supported.
- **graph parameters** - no functions, but limited support for the following:
  - areaMode - 'none' or 'all'
  - drawNullAsZero
  - hideLegend
  - height
  - lineMode - 'slope', 'staircase', or 'connected'
  - title
  - aliases (an extension of the alias parameter for legend names)
  - width

By the Graphite author's own admission (Chris Davis, an excellent article on its architecture and development here <http://www.aosabook.org/en/graphite.html>)

*"Graphite's URL API is currently a sub-par interface in my opinion. Options and functions have been tacked on over time, sometimes forming small islands of consistency, but overall lacking a global sense of consistency."*

So although it might be a bit ugly, we're going to have to stick to it if we want URL-level compatibility with third-party Graphite tools.



## DataDash

This module provides the DataYours dashboard interface, pulling together all the separate components of the system into one 'seamless' (hopefully) application - even across multiple processors. It is totally different from the dashboard interface of the Graphite system but oriented towards the Vera environment showing key features such as devices, device variables, serviceids, etc.



## DataMineServer

This module is only a very lightly modified version of a Vera utility called dmDBserver that I wrote a while ago. With just a few tweaks it fits right in to the DataYours system. The dataMine read-only database access layer dmDB was written in 2013 and remains unmodified since then.

The HTTP request handler responds to browser URLs of the form:

```
http://[your Vera IP address]:3480/data_request?id=lr_dmDB
```

The above request, on its own, responds with 'help text' describing the allowed parameters:

PARAMETERS: names and values can be abbreviated

[actions]

```
    plot = plot specified dataMine channel [number]
    graph = plot specified dataMine graph [number]
    report = various report types [channels/graphs]
```

[searchKeys]

```
    name = dataMine channel name [string]
    channel = dataMine channel id [number]
    variable = Luup variable name [string]
    devicenum = Luup device number [number]
    serviceid = Luup serviceId [number]
```

[options]

```
    height = HTML output height [number]
    format = report format [csv/iso/Table/LineChart/AreaChart]
    width = HTML output width [number]
```

```
[times]
    t2/stop/to = stop times in unix or ISO format [string]
    t1/start/from = start times in unix or ISO format [string]
    dt/interval = duration [day/week/month/year]
```

EXAMPLE: &start=2013-12-01&device=123&report=log&format=Table

There are four basic request types:

1. **searches** - these allow retrieval of raw dataMine channel data in various formats: all described here <http://forum.micasaverde.com/index.php/topic,17499.msg136838.html#msg136838>
2. **reports** - two flavours (both these types can be sorted by different columns - just click):
  - **&report=channels** - lists all the channels which dataMine is logging
  - **&report=graphs** - lists all the graphs which are defined in dataMine
3. **plots** - enables plotting of individual dataMine channels. For example:
  - **&plot=3&int=week** - plots channel 3 data over the last week (you can use day/week/month, I wouldn't advise year). This is limited to 2000 points, to save you running out of memory.
  - **&plot=3&from=2013-10-01T00&to=2013-11-01T00** - plots channel 3 for the month of October (these are ISO 8601 format dates)
4. **graphs** - recreate predefined plots, numbered by dataMine from 1 to N. Example:
  - **&graph=7** - plots whatever you have defined as graph 7
  - **&graph=7&interval=week** - as above, but overrides the default timescale to show, in this case, the last week's worth of data



## Acknowledgements

Icons by icons8 <http://icons8.com/license/>

## References

The Graphite System:

<http://aosabook.org/en/graphite.html>

<https://graphite.readthedocs.org/en/latest/overview.html>

Whisper

<https://graphite.readthedocs.org/en/latest/whisper.html>

Carbon Cache:

<https://graphite.readthedocs.org/en/latest/config-carbon.html>

---