Vhome Java Platform Description

This document presents an overview of the Vhome data model, and describes the Vhome APIs and their use.

Data Model

1. Classes

Data is organized by dividing it into 'classes', where each 'class' describes a unique type/kind of data stream.
Classes are stored in a relational database table *Class*:

Class: (cid, cname, descriptor, rating)

where,
cid : integer primary key called class identifier (> 0, unique for each class)

cname: string class name (may or may not be unique across classes, cannot be null, max length 100 characters)

descriptor: string auxiliary description of the class (may be null, max length 200 characters)

rating: a float value associated with the class, describing *wattage* of the class (may be null).

Sample entries in a *Class* table:

cid	cname	descriptor	rating
1	lighting	Philips CFL room lights in the home	100
2	dishwasher	Kenmore dishwasher	400
3	envi	Mean power measured using Current Cost Envi	NULL
4	smartPowerBoard	Zigbee Smart Power Board with five power outlets	NULL
5	utilitySmartMeter	Smart meter readings from Waterloo North hydro	NULL
6	weather	Weather characteristics by UW Weather station	NULL

Table I

2. Objects

Given the classes, each class has different objects associated with it, and each object corresponds to one time series. Objects describe and store the different time series emanating from one particular class. Hence, objects corresponding to the same class have same *cid* but different object identifiers.
Objects are stored in a relational database table *Objects*:

Object : (cid, oid, oname, descriptor, granularity)

cid: integer class identifier of the class associated with this object (foreign key, on Class).
oid: integer identifier of the object for this particular class (>0), and unique per cid value.

Hence, {cid,oid} is the composite key for the table and uniquely identifies an object.

oname: string object name (may or may not be unique across objects, cannot be null, max length 100 characters).

descriptor: string auxiliary description of the class (may be null, max length 200 characters)

granularity: integer granularity (measured in milliseconds) of the time series corresponding to this object. (Granularity is the time difference between successive entries in the time series.)

Unit: Symbol of units in which the data value is measured (for e.g. W, kWh, Wh, J, C, $).

Note that, two object entries of the same class, (same cid but different oid), may differ only in granularity and have same oname and descriptor.

In relation to class entries in Table I, sample entries in a *Object* table:

cid	oid	oname	descriptor	granularity	unit
1	1	bedroom	replaced 01/06/2011	2000	W
1	2	kitchen	replaced 01/07/2011	2000	W
1	3	garage	replaced:01/01/2011, warranty: 01/01/2020	2000	W
2	1	dishwasher	installed: 07/01/2011	5000	W
3	1	mainFeed	installed: 07/01/2011, aggregate mean power	1000	W
3	2	mainFeed	installed: 07/01/2011, aggregate energy consumption	3600000	kWh
3	3	mainFeed	installed: 07/01/2011, aggregate energy consumption	90000	Wh
4	1	socket1	powers the TV, rating: 200.0	3000	W
4	2	socket2	powers PC, rating: 190.0	3000	W
4	3	socket3	powers music system, rating: 900.0	3000	W
5	1	smartMeter	Smart meter data scraped by data scraper application	3600000	kWh
6	1	temperature	Weather data scraped by weather scraper	600000	C
6	2	humidity	Weather data scraped by weather scraper	600000	%

Table II

3. Streams

Each object (has a unique {cid, oid}) and corresponds to one stream or time series, stored as table *S-cid-oid:*

S-cid-oid : (timestamp, value)

where,

timestamp: long integer UTC millisecond timestamp, denoting the time-instant or beginning of the time-interval (e.g. beginning of the hour/minute), for the measured *value.*

value: float value of the data value at that timestamp.

Some objects may not have any timeseries data associated with them, in which case, the table S-cid-oid is empty. The timestamps and the values adhere to the granularity and unit values for the objects' entry in the Object table.

Sample entries in time-series tables are:

S-1-1

timestamp	value
1307897545000	1900.11
1307897547000	1918.12
1307897549000	1971.12

S-5-1

timestamp	value
1339473600000	0.32
1339477200000	1.98
1339480800000	2.11
1339484400000	2.91

APIs

The platform offers APIs using a RESTful service over HTTPS, which allows applications to read data (as per the model) and create new Classes and Objects, and hence store any time-series data. We now detail the RESTful service methods, their parameter types and return values.

1. ListAllClasses
Type: GET
Response: All attributes of all entries present in the Class table.
Return type: JSON array.

2. ListClass/param/value
Type: GET
Parameter :
param = either '*cid'* or '*cname'*
value = *cid* value (integer) or *cname* value (string) for the desired Class.
Response: All attributes of Class entry(or entries) with the given cname or cid.
Return type: JSON array.
e.g. ListClass/cid/1, ListClass/cname/lighting
returns complete class entry for cid=1 as a JSON array.

3. ListClass/rating/x/y
Type: GET
Parameter: floats x and y
Response: All attributes of Class entry(or entries) with *rating* $\in$ [x,y].

Return type: JSON.
e.g. ListClass/rating/100/220
returns class entries with rating in [100,220] as JSON array.

4. ListAllObjects

Type: GET
Response: All attributes of all entries present in the Objects table.
Return type: JSON array.

5. ListObject/param1/value1/param2/value2

Type: GET
Parameter :

param1 = either '*cid'* or '*cname'*
value1 = *cid* value (integer) or *cname* value (string) for the Class of desired Object.
param2 = either 'o*id'* or 'o*name'*
value2 = o*id* value (integer) or o*name* value (string) for the desired Object.

Response: All attributes of Object entry (or entries) with the given cname or cid, and given oid or oname.
Return type: JSON array.
e.g. ListObject/cid/1/oid/2, ListObject/cname/lighting/oname/kitchen
returns complete Object entry for cid=1 and oid=2 as a JSON array.

6. ListObject/param1/value1/granularity/x/y

Type: GET
Parameter :

param1 = either '*cid'* or '*cname'*
value1 = *cid* value (integer) or *cname* value (string) for the Class of desired Object.
x, y = range [x,y] for the granularity values of desired objects (with the given cid or cname).

Response: All attributes of Object entry (or entries) with the given cname or cid, and granularity $\in$ [x,y].
Return type: JSON array.
e.g. ListObject/cid/3/granularity/900000/3600000,
returns complete Object entries for cid=3 and granularities in [900000,3600000] as a JSON array.

7. ListObject/rating/x/y/param1/value1

Type: GET
Parameter :

x, y = range [x,y] for the rating values of desired class(es).
param1 = either 'o*id'* or 'o*name'* of the desired Object(s).
value1 = o*id* value (integer) or o*name* value (string) for the desired Object(s).

Response: All attributes of Object entry (or entries) with Class rating in [x,y] and given oid or oname values.
Return type: JSON array.
e.g. ListObject/rating/100/400/oname/mainFeed,
returns complete Object entries with rating in [100,400], named mainFeed, as a JSON array.

8. ListObject/rating/x/y/granularity/z/w
Type: GET
Parameter :
x, y = range [x,y] for the rating values of desired class(es).
z, w = range [z,w] for the granularity values of desired object(s).
Response: All attributes of Object entry (or entries) with Class rating in [x,y] and granularities in [z,w].
Return type: JSON array.
e.g. ListObject/rating/100/400/granularity/0/1000,
returns complete Object entries with rating in [100,400], and granularity up to 1000 mS as a JSON array.

(APIs that need explicit authorization from the user. TODO: Add more detail about their use).
9. AddClass/cname/x/descriptor/y/rating/z
Type: POST
Response: New class entry as a JSON array.

10. AddObject/cid/x/oname/y/descriptor/z/granularity/w
Type: POST
Response: New object entry as a JSON array.

Using the APIs

The Vhome RESTful service is hosted over a Apache Tomcat web-server, over HTTPS.

1. Cloud Based Apps
These apps can use the different function calls (described above) by encoding the respective function call (with parameter values) as a URI.
For e.g. https://vhomeIP/data/ListAllClasses , https://vhomeIP/data/ListClass/cname/lighting
However, they need to first obtain a token for access to the respective call(s) and present it as a POST parameter.
TODO: Add more detail.

2. Native
For providing access to native apps, the RESTful function calls described above, are wrapped into respective Java functions, thereby forming a Java library, and provided to app developers in a jar file. Native applications can directly invoke these functions without having to deal with authentication and authorization, as in case of cloud based applications.

This allows the platform to restrict native applications' use of functions in JAVA.NET.* (i.e. Dis-allowing any network access, and hence preventing native applications from transferring data out of the Vhome).

Application developers compile their Java web applications as a war file, which can then be deployed over the Apache tomcat server using a browser UI (https://localhost:8080). However, before deploying the application the platform scans the war file to ensure no use of java.net.* by the native application. After being successfully deployed the application can be invoked on the Vhome, using a browser and pointing it to *https://localhost:8080/MyNewNativeApp.*