Data Model for Residential Buildings Research

Xin Jin and Deepthi Vaidhynathan

National Renewable Energy Laboratory

Version 0.1, April 2016

Table of Contents

1. Overview 2
  1.1. Purpose 2
  1.2. Background 2
  1.3. Extension of Project Haystack 2
2. TagModel 3
  2.1. Metamodel 3
  2.2. Entities 3
  2.3. Tags 3
3. Structure of the Data Model 4
  3.1. Site 4
  3.2. Equip 5
  3.3. Point 5
  3.4. Weather 7
4. Systems and Equipment 8
 4.1. Thermostat 8
  4.1.1. Tags 8
  4.1.2. Points 9
 4.2. Water Heater 10
  4.2.1. Tags 10
  4.2.2. Points 10
 4.3. CEA-2045 Devices 11
  4.3.1. Tags 11
  4.3.2. Points 11
 4.4. Meters 12

1. Overview

1.1. Purpose

This document defines an extensible and open-source data model for residential buildings. The purpose of a data modeling standard is to provide a consistent, standardized methodology for naming and describing data elements and their relations. The data elements considered in this document are the ones associated with building energy management systems, equipment, appliances, sensors, other smart devices, and associated descriptive information[1].

1.2. Background

The data model is based on Project Haystack, an open source initiative focused on developing naming conventions and taxonomies for Internet of Things (IoT), specifically, connected building equipment and operational data[2]. Project Haystack uses a simple metamodel called *"tags"* to model building equipment and operations. Tags are name/value pairs which may be associated with an entity like a home appliance.

1.3. Extension of Project Haystack

Although Project Haystack is designed to work with the IoT on a broad scale, the current focus of Project Haystack is HVAC systems in commercial buildings. Project Haystack documents provide definitions on AHUs, VAVs, chillers, boilers. However, models of other important building equipment and appliances have not been developed. The goal of the current document is to extend Project Haystack to the area of residential buildings. With this extension, a common taxonomy is established so that residential buildings and commercial buildings can interoperate with each other to provide grid services.

To make the document self-contained, some parts of this document are adopted from the Project Haystack document. This is a living document and will be frequently updated.

[1] Guide Specification for Data Modeling of Building Systems and Equipment Based on Project Haystack Open Source Data Modeling Standard. http://project-haystack.org/download/file/Guide-Specification.docx
[2] Project Haystack. http://project-haystack.org/

2. TagModel

2.1. Metamodel

Similar to Project Haystack, a simple metamodel, *tags*, is used in this document to define the model of residential building equipment and operations. Metamodel defines the framework for the model; in other words, metamodel is the model of the model. Tags are flexible enough to build standardized models which are customized on a per-project or per-equipment basis.

2.2. Entities

An *entity* is an abstraction for some physical object in the real world. Entities include sites, equipment, sensor points, weather stations, etc. This document does not specify how entities are stored, instead it only defines how to tag those entities with specific name/value pairs. We can build a taxonomy which enables interoperability in residential building by using a library of standardized tags.

2.3. Tags

A *tag* defines a fact or attribute about an entity. Tag names should follow the conventions which are specified in Project Haystack document:

- Must start with ASCII lower case letter (a-z)
- Must contain only ASCII letter, digits, or underscores (a-z, A-Z, 0-9, _)
- By convention we use camel case in tags (e.g., geoAddr, primaryFunction)
- Use dash to connect different components of text in identifier (e.g., main-meter-kw)
- Use underscores to connect the tags or words in a point name per Google Python Style Guide[3] (e.g., hvac_heat_sp)

A *kind* is one of the permitted value types of a tag. Here is a list of commonly used tag kinds:

- **Bool**: Boolean "true" or "false".
- **Int**: Integer number annotated with an optional unit of measurement.
- **Float**: Floating number annotated with an optional unit of measurement.
- **Str**: a string of Unicode characters.
- **Ref**: reference to another entity. Use a prefix "@" followed by the referred entity.
- **Date, Time and DateTime**: a timestamp following the ISO 8601 format

The *id* tag is used to model the unique identifier of an entity using a Ref value type. The identified may be used by other entities to cross-reference using tags such as *siteRef*, *equipRef*.

The *dis* tag is used with all entities as the standard way to define the display text used to describe the entity. Dis should be succinct but fully descriptive of the entity.

[3] Google Python Style Guide. https://google.github.io/styleguide/pyguide.html

3. Structure of the Data Model

The primary structure of the data model is based on a hierarchy of three entities:

- Site: single building with its own street address
- Equip: physical or logical piece of equipment within a site
- Point: sensor, actuator or setpoint value for an equip

Other core entities include:

- Weather: outside weather conditions

The following diagram illustrates this three-level hierarchy and how they cross-reference each other:

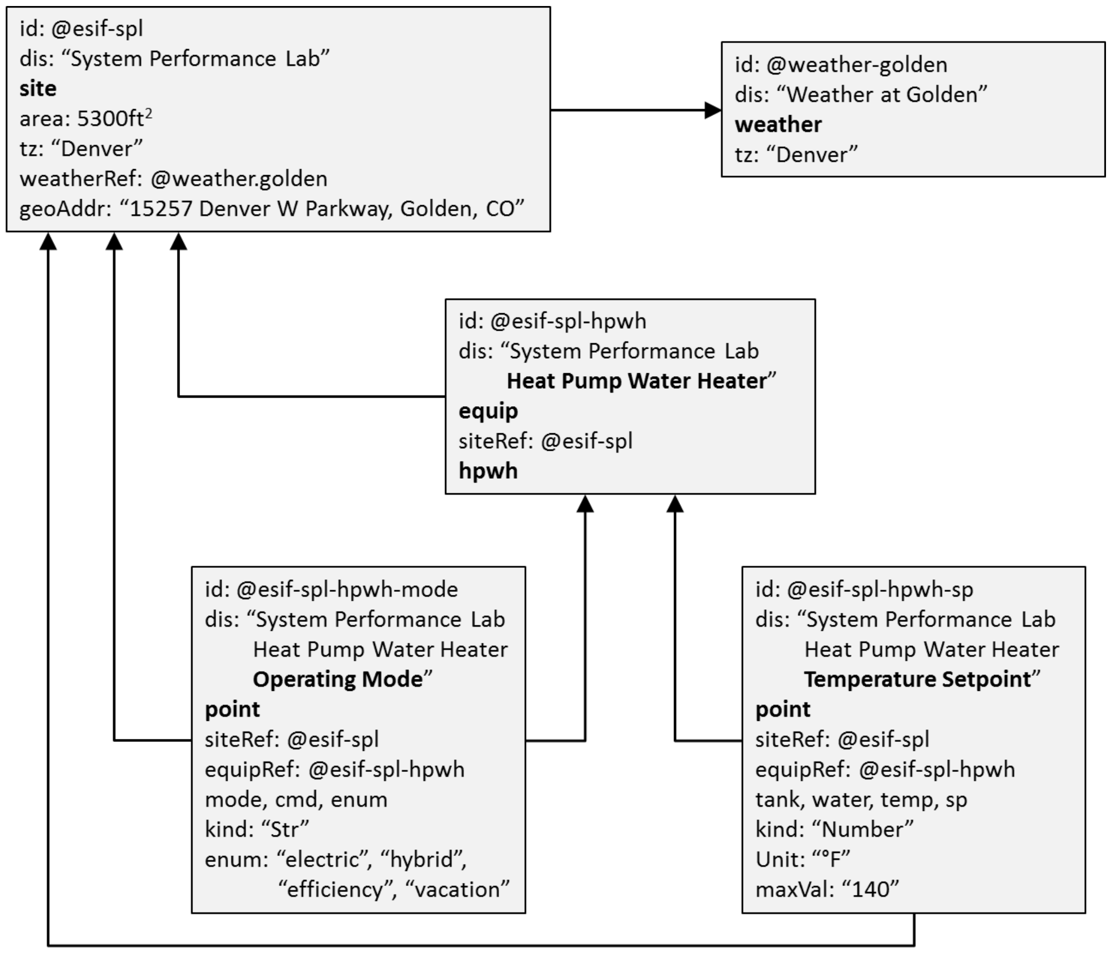

Figure 1. An example of the hierarchical data model structure: site, equip, point and weather

3.1. Site

A site entity models a single facility using the site tag. Core tags used with sites:

- **geoAddr**: the geographic free-form address of the site
- **tz**: the timezone where the site is located
- **area**: square footage or square meters of the facility

- **weatherRef**: associate the site with a weather station to visualize weather conditions
- **primaryFunction**: enumerated string which describes the primary function of the building

Here is an example of the site entity which is also shown in Figure 1:

```
id: @esif-spl
dis: “System Performance Lab”
site
area: 5300ft²
tz: “Denver”
weatherRef: @weather.golden
geoAddr: “15257 Denver W Parkway, Golden, CO”
```

3.2. Equip

Equipment is modeled using the *equip* tag. Equipment is often a physical asset such as an air conditioner, water heater, or refrigerator. However, *equip* can also be used to model a logical grouping such as a HVAC system.

All equipment should be associated within a single site using the *siteRef* tag. In turn, equipment will often contain points which are associated with the equipment via the *equipRef* tag.

Here is an example of a heat pump water heater entity which is also shown in Figure 1:

```
id: @esif-spl-hpwh
dis: “System Performance Lab Heat Pump Water Heater”
equip
siteRef: @esif-spl
hpwh
```

3.3. Point

Points are typically a sensor or actuator entity. Points can also represent a configuration value such as a setpoint or schedule log. Point entities are tagged with the point tag.

All points are further classified as *sensors*, *commands*, or *setpoints* using one of the following three tags:

- sensor: input, physical sensor
- cmd: output, actuator, command
- sp: setpoint, internal control variable, schedule

All points must be associated with a site via the *siteRef* tag and a specific piece of equipment via the *equipRef* tag. By convention multiple tag are used to model the role of a point:

- where: discharge, return, exhaust, outside, inlet, outlet, tank
- what: air, water, steam
- measurement: temp, humidity, flow, pressure

Here is an example of temperature setpoint of a heat pump water heater which is also shown in Figure 1:

```
id: @esif-spl.hpwh.sp
dis: “System Performance Lab HPWH Temperature Setpoint”
point
siteRef: @esif-spl
equipRef: @esif-spl.hpwh
tank, water, temp, sp
kind: “Number”
unit: “°F”
maxVal: “140”
```

Note that a maximum value, 140°F, is imposed on the temperature setpoint to ensure safety. Here is another example of a water heater operating mode which is also shown in Figure 1:

```
id: @esif-spl.hpwh.mode
dis: “System Performance Lab HPWH Operating Mode”
point
siteRef: @esif-spl
equipRef: @esif-spl.hpwh
tank, water, cmd, enum
kind: “Str”
enum: “electric”, “hybrid”, “efficiency”, “vacation”
```

Different from the previous example, the operating mode in this example is a “Str”, which models an enumerated point with an operating mode such as “electric”, “hybrid”, “efficiency”, or “vacation”.

3.3.1. Point Kinds

Points are classified as Bool, Number, or Str using the *kind* tag:

- **Bool**: model digital points as true/false.
- **Int**: model discrete state or .
- **Float**: model measurements or setpoints such as temperature or pressure. These points should also include the *unit* to indicate the point’s unit of measurement.
- **Str**: models an enumerated point with a mode such as “Off, Slow, Fast”. Enumerated points should also define an *enum* tag.

3.3.2. Point Min/Max

The following tags may be used to define a minimum and/or maximum for the point:

- minVal: minimum point value
- maxVal: maximum point value

When these tags are applied to a *cmd* or *sp*, they model the range of valid inputs when commanding the point. Similarly, when applied to a *sensor* point, they model the range of values the sensor can read and report.

3.3.3. Point Cur

The term *cur* indicates synchronization of a point's current real-time value. If a point supports a current or live real-time value then it should be tagged with *cur* tag.

3.3.4. Point Write/Read

Writable points are points which model an output or a setpoint and may be commanded. Writable points should be tagged with the *writable* tag.

Similarly, readable points are points which model an input or a setpoint and may be accessed. Readable points should be tagged with the *readable* tag.

A point can be tagged with either or both of the *writable* and *readable* tags.

3.4. Weather

Building operations and energy usage are heavily influenced by weather conditions. The *weather* tag models a top-level entity that represents a weather station or logical grouping of weather observations.

Weather data follows the same convention as points, but they are associated with a weather entity instead of a site entity. Therefore, we use the special *weatherPoint* to indicate a weather related point.

The following weather points are defined by the standard Project Haystack library:

- weatherCond: enumeration of conditions (clear, cloudy, raining)
- drybulb_temp: dry bulb outdoor temperature in °C or °F
- wetBulb_temp: web bulb outdoor temperature in °C or °F
- humidity: percent humidity

Weather points are associated with their weather entity using the *weatherRef* tag.

4. Systems and Equipment

Figure 2 shows an example of the data model that consists of multiple systems and equipment. The definitions of these systems and equipment are explained in this section. This section is not meant to provide an exhaustive list of systems and equipment for a typical residential building.

The Home Energy Management System (HEMS) is not explicitly modeled in this section because all the associated points have been defined in other equipment.

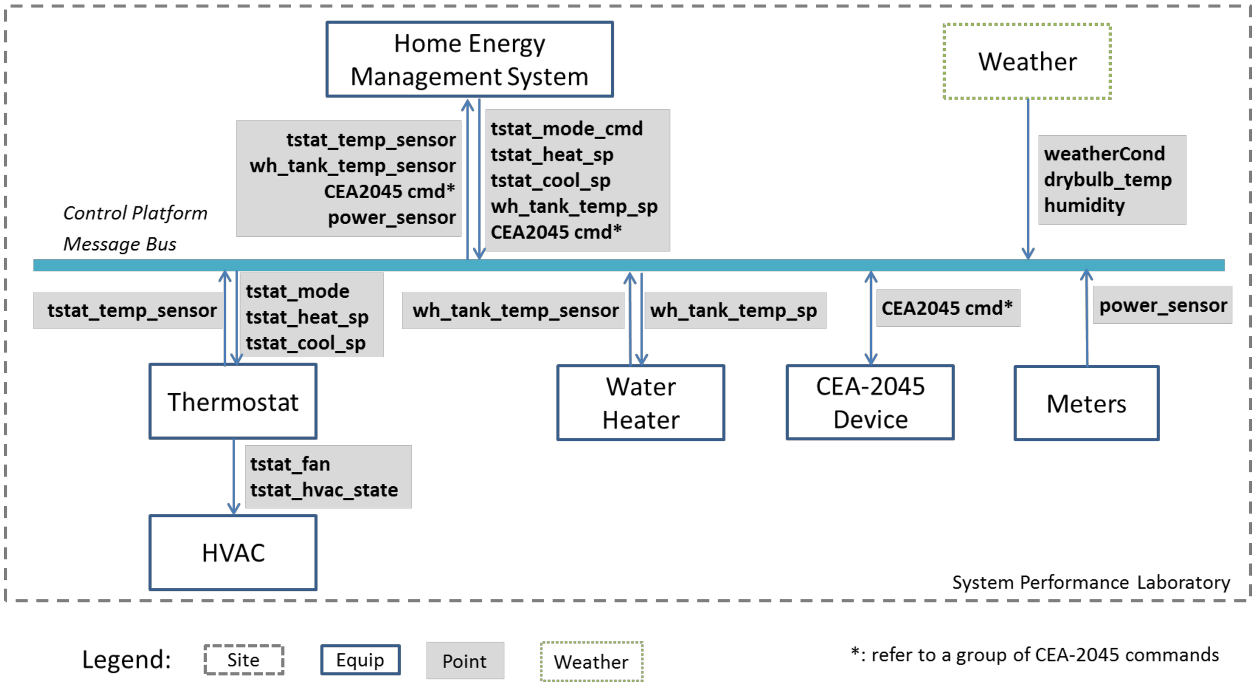

Figure 2. An example of data model for a typical residential building configuration

4.1. Thermostat

Most residential HVAC systems are controlled by thermostats. Typical central HVAC systems in residential buildings include an air conditioner, a blower fan and a heater (furnace, baseboard heater, or air/ground-source heat pump). Thermostats control the HVAC components based on the measured room temperature, the setpoint and the mode.

4.1.1. Tags

Thermostat should be marked with the *tstat* tag as well as *equip*. The points associated with the HVAC components can also be marked with the *hvac* tag. The thermostat and HVAC can be nested via the *equipRef* tag.

4.1.2. Points

Based on the Radio Thermostat API[4], the following points are used to model a thermostat:

- tstat_mode
  - dis: “Thermostat operating mode"
  - tstat, mode, cmd, writable, readable
  - kind: “Int”
  - enum: “0: Off”, “1: Heat”, “2: Cool”, ”3: Auto”
- tstat_temp_sensor
  - dis: “Thermostat current temperature reading”
  - tstat, air, temp, sensor, readable, cur
  - kind: “Float”
  - unit: “°F”
  - minVal: 20
  - maxVal: 120
- tstat_heat_sp
  - dis: “Thermostat heat setpoint”
  - tstat, air, temp, sp, writable, readable
  - kind: “Float”
  - unit: “°F”
  - minVal: 45
  - maxVal: 99
- tstat_cool_sp
  - dis: “Thermostat cool setpoint”
  - tstat, air, temp, sp, writable, readable
  - kind: “Float”
  - unit: “°F”
  - minVal: 45
  - maxVal: 99
- tstat_fan_mode
  - dis: “Fan operating mode”
  - tstat, hvac, temp, cmd, writable, readable
  - kind: “Int”
  - enum: “0: Auto”, “1: Auto/Circulate”, “2: On”
- tstat_hvac_state
  - dis: “HVAC operating state”
  - tstat, mode, cmd, readable
  - kind: “Int”
  - enum: “0: Off”, “1:Heat”, “2: Cool”

[4] Radio Thermostat Company of America, Wi-Fi USNAP Module API, Version 1.3, March 22, 2012. Available on http://lowpowerlab.com/downloads/RadioThermostat_CT50_Honeywell_Wifi_API_V1.3.pdf. Retrieved on April 6, 2016.

4.2. Water Heater

Water heaters can be classified by the heat sources. Electric water heaters, which are the primary focus of this document, can be divided into electric resistive water heaters (ERWH) and heat pump water heaters (HPWH).

4.2.1. Tags

Water heaters should always be marked as *equip* and *wh*. Depending on the heat sources, another tag *erwh* or *hpwh* should also be used to identify the types. The heat pump inside a HPWH can also be modeled as sub-equipment and the standard *heatPump* tag can be applied.

4.2.2. Points

A water heater usually includes the following points:

- wh_tank_temp_sensor
    - dis: “Water heater tank temperature measurement”
    - wh, tank, water, sensor, cur, writable, readable
    - kind: “Float”
    - unit: “°F”
    - valMin: 60
    - valMax: 140
- wh_tank_temp_sp
    - dis: “Water heater tank temperature setpoint”
    - wh, tank, water, sp, writable, readable
    - kind: “Float”
    - unit: “°F”
    - valMin: 60
    - valMax: 140
- wh_mode
    - dis: “Water heater operating mode”
    - wh, cmd, enum, writable, readable
    - kind: “Str”
    - enum: “electric”, “hybrid”, “efficiency”, “vacation”
- wh_state
    - dis: “Water heater operating states”
    - wh, cmd, enum, readable
    - kind: “Int”
    - enum: “0: off”, “1: upper element running”, “2: lower element running”, “3: heat pump running (HPWH only)”
- wh_control_method
    - dis: “Water heater control methods”
    - wh, cmd, enum, readable, writable
    - kind: “Int”

    - enum: “0: remote control disabled, use local setpoint”, “1: remote control of setpoints”, “2: remote control of heating elements”

Experimental prototypes may have additional points, such as:

- wh_inlet_water_temp_sensor
- wh_outlet_water_temp_sensor
- wh_outlet_water_flow_sensor
- wh_exhaust_air_temp_sensor (for HPWH only)

4.3. CEA-2045 Devices

The ANSI/CEA-2045 standard (now ANSI/CTA-2045 standard[5]) defines a universal modular communication interface (MCI) for energy management[6]. It specifies both the hardware interface and the communication protocol. The CEA-2045 utilizes the RS-485 and Serial Peripheral Interface (SPI) which are supported by most microcontrollers today. The MCI protocol is capable of passing through standard protocols including TCP/IP, OpenADR, and SEP from the communication module to the end-device. The CEA-2045 devices considered in this section may either be a universal communication module (UCM) or a smart grid device (SGD).

4.3.1. Tags

CEA-2045 devices should be marked with an *equip* tag. Reference tags such as *equipRef* and *siteRef* should also be specified wherever applicable. The following tags only apply to CEA-2045 devices.

- dataLink: data-link messages
- basicDR: basic DR application messages
- intermediateDR: intermediate DR application messages

4.3.2. Points

All points of a CEA-2045 device are marked with tag cmd and tag kind “Str”. Commonly used CEA-2045 points for a UCM include:

- data_link_response
    - dis: “Command for sending data-link layer response”
    - cea2045, cmd, dataLink, writable
    - kind: “Str”
    - enum: “link_ack: \x06\x00”,
      “link_nak_checksum_error: \x15\x03”,

[5] The Consumer Electronic Association (CEA) changed its name to Consumer Technology Association (CTA) and the name change applies to the standards issued by this organization. The old name of the standard (CEA-2045) is used in this document instead of the new name (CTA-2045).
[6] ANSI/CTA-2045: Modular Communication Interface for Energy Management. Available at https://www.cta.tech/Standards/Standard-Listings/R7-8-Modular-Communication-Interface-for-Energy-Ma/CEA-2045.aspx

"link_nak_message_timeout: \x15\x05",
"link_nak_unsupported_msg_type: \x15\x06"

- supported_msg_type
  - dis: "Command for querying supported message type"
  - cea2045, cmd, datalink, writable
  - kind: "Str"
  - enum: "query_supported_msg_data_link: \x08\x03\x00\x00\x76\xD3",
    "query_supported_msg_basic_dr: \x08\x01\x00\x00\x7E\xCD",
    "query_supported_msg_intermediate_dr: \x08\x02\x00\x00\x7A\xD0"
- max_payload_size
  - dis: "Query maximum payload size"
  - cea2045, cmd, datalink, writable
  - kind: "Str"
  - enum: "send_max_payload_length: \x08\x03\x00\x02\x19\x07\xA9\x7E"
- customer_override
  - dis: "acknowledge successful receipt and support of previous command"
  - cea2045, cmd, basicDR, writable
  - kind: "Str"
  - enum: "customer_override_ack: \x08\x01\x00\x02\x03\x11\xE3\x52"
- comm_status
  - dis: "Sent from UCM to SGD when the outside communication status is gained or lost"
  - cea2045, cmd, basicDR, writable
  - kind: "Str"
  - enum: "comm_status_good: \x08\x01\x00\x02\x0E\x01\xE2\x58",
    "comm_status_poor: \x08\x01\x00\x02\x0E\x02\xE0\x59",
    "comm_status_lost: \x08\x01\x00\x02\x0E\x02\xE4\x57"
- cea2045state
  - dis: "Commands for querying and changing operating states"
  - cea2045, cmd, basicDR, writable
  - kind: "Str"
  - enum: "query: \x08\x01\x00\x02\x12\x00\xD8\x5F",
    "shed: \x08\x01\x00\x02\x01\x00\x0C\x3D",
    "normal: \x08\x01\x00\x02\x02\x00\x09\x3F",
    "critical_peak_event: \x08\x01\x00\x02\x0A\x00\xF0\x4F",
    "emergency: \x08\x01\x00\x02\x0B\x00\xED\x51",
    <custom>: cea2045state commands with specified time duration in Opcode2

4.4. Meters

Meters are modeled as *equip* entities with the *meter* tag. Despite the large number of meter types, the following meter types are considered in this document:

- elec meter
- water meter

In order to model submeters and their relationships with the main meters, all meters must also define one of these two tags:

- siteMeter: marker applied to the site-level meter
- submeterOf: Ref to parent meter

Electric meters are the most common meter type. The following points are used to model an electric meter in addition to the *elect meter* tags:

- power sensor
- energy sensor
- current sensor
- voltage sensor
- pf sensor
- frequency sensor

Water meters may refer to any meter which measures the property of water. The following points are used to model a water meter in addition to the *water meter* tags:

- flow sensor
- volume sensor
- temp sensor
- pressure sensor