Schoten - the really smart home implementation with Lucy

Lucy is the smart home software that delivers an integrated experience for all home control purposes. It is voice controlled by Amazon Alexa, Ok Google and Apple Siri and drives many interfaces such as Lutron, Philips Hue lights, Ikea Tradfri and vera control ltd Vera.
My house in Schoten is implemented with this Lucy, and these site implementation files are stored here. These are private to the owner of the house and as much serve as an implementation example of Lucy as a full functional house solution.
There is just 1 configuration file that define a site implementation, `site.conf`.
This readme file covers the structure of this file and has click through to all the underlying support documents.

site.conf : the Portable Home Configuration File (PHCF)

Config Parser Structure

This file is a configuration file that contains all the home definitions and descriptors. It implements a basic configuration language which provides a structure similar to what's found in Microsoft Windows INI files. This file can be customized by end users if they have some computer knowledge and text file editing skills.
For more information on config parsing, see https ://docs.python.org/3/library/configparser.html
The syntax of site.conf is checked by a config parser program by every raspberry node in the network and is the basis of the data structures used by Lucy's scripts and an extensive html report is available once parsing is completed.
The file app_obj.conf in the Lucy package (not the home package) contains all the fields and their purpose for every interface or control. The config parser reads this definitions to validate the parsing of site.conf, so it is the best source for information and validation.
The first 3 sections (square braced words in the beginning of the line) in site.conf are mandatory and should be in that order, then the rooms or places are defined (in square braces) :

1. **[DEFAULT]** : do not change, this is a config parser thing
2. **[VERSION]** : version of the config file, update this after making changes
3. **[APPS]** : the system apps and the drivers, the functional apps can only be defined in the room or place where they provide their functionality
4. __[ROOM/PLACE]*__ : the sections where rooms (in-house) or places (out-house) definitions are specified

Comments must be prefixed with # or ;
Example (... means must continue) :

```
[DEFAULT]       # is applied automatically to every room
    is_room=True
    security={"zone":"house"}
[VERSION]       # should be first, is mandatory and contains the version information of this file
    is_room=False
    version="h-3.0.0"
[APPS]    # should be after version, is mandatory and contains all non room linked configuration, interfaces and dr
    is_room=False
    site_settings={
            "latitude":             51.2849951,                 # of your location.  It is used for sunrise / sunse
            "longitude":            4.5322251,                  # -180..180
            "currency":             "€&euro;",                  # 1 to 3 characters the currency letters and the HT
            "degrees":              "°C",                       # °C or °F
```

```
            # **** Below are the special status virtuals, is_holiday does not imply that that alarm system is armed
            "is_reboot":            Virtual("app", duration=1),   # do not change the duration
            "is_day":               Virtual("app", value_logic={"00:00":"False","sunrise+00:15":"True","sunset-00:1
            "EMAIL_prime":          "rudy_vandenberghe@hotmail.com",
            "EMAIL_other":          "lucyaraujo13@hotmail.com",
            "language":             "english", # not yet implemented
            "IP-WAN":               "8.8.8.8", # not yet implemented
            "site_id":              "schoten"}
    ...
[STREET]
...
[GARDEN]
...
[KITCHEN]
...
```

Firstly, the core apps are outlined, followed with the system apps, the drivers and then the functional apps. Then base things follow which are the basis of all things which are presented last.
Potentially you can skip the core apps and first visit the rest, and come back to these if you need more detail.

Core Apps

what_is	Description
(docs/config_parser.md)	Config Parsing verifies and turns your site.conf file into program data
(docs/config_parser.md)	Multiple Occurrences for complex functional apps such climate_system or security_system or
(docs/Notifications.md)	Notifications are triggered by an app or a thing and post their message through different cha
(docs/pin.md)	the reference to the physical aspect of a thing, such as an input, an output or sensors
(docs/properties.md)	more detailed information on item properties, adding features and logic
(docs/room_place.md)	define rooms, subrooms, places and zones and link this all together
(docs/devices.md)	definable input things or output things
(docs/things_sync.md)	The process of ensuring that the as_is and to_be state of a Thing is there where it needs to b
(docs/value_logic.md)	value logic : value setting/blocking based on conditions or time events
(docs/Virtual.md)	Virtuals are Virtual Things and they make your configuration smart and easy to understand

Multi App occurrences

To allow multiple instances of security and climate system, the options can be indexed by adding an index trailer : _x with x a number and add to the raspberry roles a keyword 'option_x' with x the index number, so that only these options will be parsed for the raspberry that will handle that set of security or climate arrangements.
Example of 2 climate systems managed by 2 separate raspberry's :

```
climate_system_0={"role_me":"PI-Climate",..}
climate_system_1={"role_me":"PI-Climate2",..}
raspi={
    "PI-Climate":  Raspi(ip="192.168.15.70",io_dev="UniPi,24,14",color="magenta", roles=["option_0"]),
    "PI-Climate2": Raspi(ip="192.168.15.71",io_dev="UniPi,24,14",color="magenta", roles=["option_1"]),
}
```

This works also for other interfaces such as irrigation, weather, etc..

Properties : adding features and logic

Properties is a keyword allowed in the definition of Inputs, Outputs, Lights, Buttons, Dimmers,.. to define a dictionary of additional attributes. The fields of this dictionary are checked for eligibility and it can be best showed through some examples.
Simple Example to have a voice message when internet access is lost or restored :

```
"internet_lost":
    Virtual("app", properties={
        "say_active":"tts_internet_lost",
        "say_normal":"tts_internet_there"})
```

A more complex example involves the use of virtuals for post mail handling.
It will trigger a voice and mail message when new mail arrives and create an email message when the mail is removed. At the same time a virtual ”have_mail” is maintained and set when there is new mail and lowered when the mail is taken.

```
mailbox_monitor={  # It is connected to 2 magnetic switches which register incoming and retrieved post mail
    "role_me": "PI-Mail",
    "mail_in": Input(pin="pi:PI-Mail,0", properties={
        "say_active":   "tts_mail",
        "have_mail":    Virtual("do_me_up"),
        "nty_active":   "nty_mail_in"}),
    "mail_out": Input(pin="pi:PI-Mail,1", properties={
        "have_mail":    Virtual("do_me_down"),
        "nty_active":   "nty_mail_out"})}
```

Nesting of properties is allowed to a few levels deep, but it is best to keep nesting levels shallow.
see a gate entrance door example for a complex nesting structure and the extensive use of virtuals and notifications.
Although this may look like programming, you can easily add features without changing any of the programming scripts.
Please study the site.conf example for tricks and features you may need.

Virtuals and Events : make your configuration smart

When pondering the strategy to separate home specific configuration from the programming scripts, the idea emerged of using virtuals and events to make the configuration smart and to have a powerful design mechanism to have logic in the site.conf configuration file. It worked out better than i anticipated and the result is a very powerful combination. The negatives, cluttering the configuration file with virtuals has proven to be manageable although the syntax of closing parentheses and braces can drive you almost nuts. However, it clearly delivers the result of having stable and separated processing code.
as an example of frequently used virtual actions (many more are available) :
Further reading on how to use virtuals.
Further reading on how to use events.

Value_Logic : set item values based on conditions or time events

Home automation results in devices getting values based on very complex circumstances and value_logic is a reasonable simple solution to capture this complexity in a (hopefully) readable manner.
Two situations can be defined, one whereby the device get overruled and a certain value is set or the other situation whereby the value of the device is frozen until this lock is lifted.
Value logic is a value setting dictionary that can be defined in output derived devices, see value_logic
A few value_logic examples will clarify :

1. below is_day is defined as True after sunrise and before sunset else False

```
"is_day": Virtual("app", value_logic={"00:00":"False","sunrise":"True","sunset":"False"})
```

2. here the curtains open 15 minutes after sunrise and before sunset or 5pm whatever comes first and freezelist the item the rest of the time.

```
"vera_curtains": Virtual("do_me_follow",
    value_logic={"sunrise+00:15":True,"00:00":False,"07:30":True,"17:00":False,"sunset":False})
```

3. this example sets the ventilation speed based on the occupancy modes of the house if it is day or night

```
"ventilation_speed": Dimmer(pin="pi:PI-WWaste,0",
    value_logic={"is_holiday":"0.10","is_armed_full":"0.10","is_armed_partial":"0.20","is_day":"0.50"})
```

4. this example sets the hall light off when nobody is there, if sleeping then the value is 10 else 25 before sunset else 100

```
"hall_light": Dimmer(pin="pi:AR-Hall,0",
    value_logic={"is_armed_partial":"10","is_armed_full":"0","is_holiday":"0","00:00":"25","sunset":"100"})
```

5. for philips hue items, a hue defined scene name can be used in the value_logic, see below example where the light will be off when the room is protected or before sunrise. Then it will be at spring_blossom at 20 (values are 0..255) till noon, then tropical_twilight at 150 and finally from sunset savanna_sunset at 50.

```
"office_ledstrip": Dimmer(pin="hue:Hue_Bridge,{:}",
    value_logic={"is_room_secure":"0",
            "00:00":"0",
            "sunrise":"20,spring_blossom",
            "12:00":"150,tropical_twilight",
            "sunset":"50,savanna_sunset"},
    properties={"type":"RGB"})
```

Referencing Things

Firstly what are input or output derived objects and what are allowed pin types?
In all input or output derived objects, the definition of a **pin=** is mandatory.
Pin types define the source of the input / output element and they have each to follow a specific syntax :
For tcp and udp, 'ip_txt' string can contain python string formatting, whereby the variables are {name} for substituting the device name and {val} the current value of the device.
Pin definition Examples :

```
"iButton_out_red":     Light(pin="pi:AR-1W-OUT,1",duration=2)          # pin is an arduino, output pin 1
"door_kitchen_terras": Door(pin="1w:A,12742ECC000000B4")               # pin is DS2406 pin A
"HVAC_test":           Clim_SP(pin="zw:p_switch,135,1",i_make={"wind"}) # pin is zw device 135 variable 1
"office_wall_light":   Light(pin="vera:{:}")                          # pin is vera device named office_wall_ligh
"guest_light":         Dimmer(pin="ikea:Ikea_Tradfri,{:}")            # pin is tradfri device guest_light
"bed_candle":          Dimmer(pin="hue:Hue_Bridge2,{:}")              # pin is on Hue Bridge 2, named bed candle
"temp_outside":        Sensor(pin="289574D906000075")                 # pin is 1w temp sensor
"tcp_light":           Light(pin="PI-Light/tcp:192.168.15.76,5000,'{name}-{val}'") # pin is tcp message
"udp_light":           Light(pin="PI-Light/udp:192.168.15.76,5000,'{val}'")        # pin is udp message
```

Notifications

Using a prefix in the properties of an option or an io item, certain notifications can be defined.
As example, in irrigation there is an attribute nty_all, that has a list of names that are notifications that will be called automatically when that particular irrigation event happens.
With a prefix, these notification triggers can be activated in PI_Home.conf in the option irrigation to allow the following :
So to make a speech notification for the completion of irrigation, the identifier ”tts_irr_completed” should be referenced in PI_Home.conf in irrigation, and ”tts_irr_completed” should be defined in PI_Install.conf, section [TTS] in the tts_msg dictionary.
Some examples :

- for range devices we have val_x (x the value), example whereby a text message is played through sonos when climatisation switches from heating to cooling :

```
"climate_mode":
    Flag_R("role",[-1,0,1], ["Cooling","Off","Heating"]),
        properties={
            "tts_val_-1":   "tts_now_cooling",
            "tts_val_1":    "tts_now_heating"})
```

- for binary devices we have active and normal. In this example there is a zwave multi switch device that get triggered when the flag do_sonos gets active and a tcp and udp message when this flag becomes non active.

```
"do_sonos":
    Flag("role", value_logic={"is_armed":"False","is_holiday":"False","00:00":"False","09:15":"True","22:30":"False
        properties={
            "zw_active":    "zw:p_switch,135,1",    # this is the vera virtual switch Voice Anotations kept in sync
            "zw_normal":    "zw:p_switch,135,0"}),
            "tcp_normal":   "tcp:192.168.15.58,5000,'text!!'",
            "udp_normal":   "udp:192.168.15.58,5000,'text!!'"})
```

- for sensors we have high/low and normal. In this example there is a notification email and a sonos speech message when the temperature is high and back to normal.

```
"Case°C_aslave_rear":
    Sensor("28FF7C1965040004",
        properties={
            "check_event": Event("check_float",
                parameters={
                    "low":  None,
                    "high": 40.0,
                    "nty_high":    "nty_box_high",
                    "nty_normal":  "nty_box_normal",
                    "say_high":    "tts_box_high",
                    "say_normal":  "tts_box_normal"})})
```

see notifier for more details on the notification syntax.

app_data.conf : the site install configuration file

This configuration file is home site specific but contains information that is typically only for the installer of the system.
Sections of this file :