The sentient Home

My dream always have been to live in a sentient home. A sentient home symbolises the future of smart homes where the house will be an entirely intuitive environment that manages external and internal necessities for residents. A sentient home has smart devices and is totally connected to its residents through voice interaction, wireless detection and identification of residents and visitors. The home will automatically adjust in real-time, a spectrum of facilities that support individuals, their preferences and copes with efficient use of utilities such as energy and water in combination with weather patterns. On a whole, these smart houses are totally automatic that the house routines on their own. Automatic applications controlled with artificial intelligence will cope with the day by day functions of the home efficiency and in a manner that accrues data so that machine learning can continue to optimise all processes.

To achieve this vision, a first step is to ensure a layer of home control that allows an universal use and control of things and a language to store preferences, and behaviour for these things.

Enter the world of universal home control.

Universal Home Control

Imagine that you have the best and latest that home control can offer in terms of comfort, and that it keeps up to date with the changing world of things. You can buy the things from the supplier(s) you want and simply make them known to your home controller. You have home apps, just like on your smartphone to configure your needs, your notifications and your wishes so that the home controller knows what it must do.

Your supplier places a modem alike box in your house, and they keep the home control running and do the updating. With this box, you are not dependent on the internet to have your home control working, and it works fast too as everything is local to deliver new payloads to your things.

And your supplier offers all kinds of cloud app goodies such as energy management, predictive maintenance, full data forensics of what happened in your home just to name a few benefits. You do not only get comfort, but you also get significant cost saving in energy and utility usage. If you are the home user, not the home owner, then the things of the owner or of the building work just if they were yours.

Your settings are separated from the services and changes and privacy is protected by blockchain technology, only you can authorize changes to your things and wishes. You determine the supplier you want for the services you require, no vendor has a lock in.

And you can leverage on what you have today and already get rid of all the vendor legacy maintenance issues you might already have.

This (not imaginary) universal home control solution depend on 3 interacting elements:

1. Things : lamps, switches, sensors, outputs, cameras, sound systems but also climate systems, security systems, electricity and water meters, ...
2. Apps : give the things behavior and logic, there are things apps (apps working on one thing), core apps (to make everything work) and functional apps such as climate management in your home
3. Drivers: plug ins to recognize Things from various suppliers or sources

These 3 elements together define your specific Home Configuration and this data is stored in a file called Portable Home Definition File (PHDF).

And just like a PDF file, this file is portable and just is everything there is to know to make it all work for you the way you want to.

And your home experience can be extreme with voice control such as Amazon Alexa, Apple Siri or Ok Google, or just very basic. But your cost savings for energy and predictive maintenance will be just the same.

Lets go into the subject of Portable Home Definition File, so that you get a feel for what it is.

But first let me introduce myself, i am Rudy Vandenberghe, an independent software developer and i created Lucy which delivers an integrated experience for all home control purposes. I am voice controlled by Amazon Alexa, Ok Google and Apple Siri and i utilize many drivers with system and things from Lutron, Philips Hue lights, Ikea Tradfri and vera control ltd Vera. Lucy is working in my house and this working model is selected as a basis of the examples in this readme file.

Portable Home Configuration File (PHCF)

This configuration file uniquely defines a site operation, **myproject.py** and this is a python3 script file.

It can be broken down in 2 parts:

1. Apps with settings in use at the site, and the drivers that link to the things
2. Rooms or places, and the things they behold and their settings, behaviors and notifications

This file is parsed by regular python3, checking and validating this file and converting the contents to apps and controller ready objects.

Currently, no app exist yet to create and maintain a PHCF, but to make one is fairly trivial. Today, a text editor can be used to create an maintain this file, and the compiler is used to check this input before translating this further into working things_controllers.

This setup allows rapid prototyping and testing, but a final product should contain a very user-friendly configuration app with wizards and guided input.

Anyway, let's start to explore a bit this configuration file, and then zoom into the different [APPS], some concepts to gather understanding and we close with a short briefing of some of the apps.

Enjoy reading!

Config Parser

myproject.py is a configuration file that contains all the home definitions and descriptors. It implements a complete python3 script which provides a structure similar to any of the many python3 script that exist with the exception that it only contains a single expression with all your home's configuration. This file can be customized by software designers and users who have some computer knowledge and text file editing skills.

The syntax of myproject.py is checked by the python3 compiler by every intelligent things controller node in the network and is thus the basis of the data structures used by Lucy's scripts. An extensive html report is available once parsing is completed. A document generator is part of the prj_parser, whereby more than 70% of the documentation is auto generated from the programming scripts.

PHDF Structure

```
from lucy import *
root_str={
    "project":Project(
        settings=[],
        drivers=[],
        apps=[],
        property=House(
            places={},
            owners=[],
            )
        )}
```

So your whole configuration is an object Project() with a list of drivers and apps and a property House() that contains all the places and the apps for these places.

A property does not need to be a house but can also be an Apartment, a Building, a Business or just some places.

Every object has it defining keywords, and sometimes objects are placed in a list as for apps and drivers and sometimes they are named in a dictionary as for project and places.

Drivers enable the controllers and things of their kind. As example enables the daikin_driver the definition of daikin thingscontrollers and daikin things.

Apps come in different types, system apps make the system work like the Network_controller() app, or functional apps like Climate_system() that regulate the climate in your rooms and then the apps that are bound to a place such as Lights() and Climate()

Example:

```
from lucy import *

settings = [
    Site_settings(
        EMAIL_groups = {
                "{everyone}":["rudy_vandenberghe@hotmail.com","lucyaraujo13@hotmail.com"],
                "{other}":["lucyaraujo13@hotmail.com"],
                "{prime}":["rudy_vandenberghe@hotmail.com"]},
        EMAIL_other = "{mail_other}",
        EMAIL_prime = "{mail_prime}",
        THINGS_account = {
                "AuthPass":"XXXXX",
                "AuthUser":"rudy.vandenberghe@gmail.com",
                "mailhub":"smtp.gmail.com:587",
                "useSTARTTLS":"YES"},
        currency = "€&euro;",
        degrees = "°C",
        language = "English.eng",
        latitude = 51.2849951,
        longitude = 4.5322251,
        site_id = "ENG"),

    Home_settings(
        home_occupancy = Virtual_R(description_r = "Day,Away,Sleep,Holiday",digital_range = "0,1,2,3",to_do = "app"),
        is_day = Virtual(to_do = "app",value_logic = {"assign":{"00:00":"False","sunrise+00:15":"True","sunset-00:10":
        is_holiday = Virtual(
                copy_things = {
                        "twin_copy":Output(pin = "zw:{tc},buttonset,152,Status1")},
                notifications = {
                        "cal_active":{"txt":"Holiday Mode Started"},
                        "cal_normal":{"txt":"Holiday Mode Stopped"},
                        "mail_active":{"subject":"Holiday Mode Started","to":"{prime}"},
                        "mail_normal":{"subject":"Holiday Mode Stopped","to":"{prime}"},
                        "say_active":{"txt":"{tts_start} holiday occupancy setting is activated{tts_end}"},
                        "say_normal":{"txt":"{tts_start} holiday occupancy setting is deactivated{tts_end}"}},
                to_do = "app"),
        is_reboot = Virtual(duration = 1,to_do = "app"),
        role_me = "PI-Security"),
```

```
    Things_additions(
        alt_name = {"descr":"thing alternative name, such as a label for a one wire sensor","repr":"str"},
        owned_by = {"descr":"the owner of the thing, must be in the validation list 'things_owners'","repr":"str"},
        things_owners = ["home_owner","home_user","building","city","gov"],
        usage = {
                "descr":"the usage of the thing, must be in the validation list 'usage_types'."+
                         "The definition is either {'watts':10} or something like {'type':'Water', 'Unit':'L/min','Qty
                "repr":"data_dict"},
        usage_types = ["watts","Water"])],

apps = [
    Network_controller(
        IP_WAN = "127.0.0.1",
        gateway = "192.168.15.1",
        internet_lost = Virtual(
                notifications = {
                        "say_active":{"txt":"{tts_start} the internet connection is down{tts_end}"},
                        "say_normal":{"txt":"{tts_start} the internet connection returned{tts_end}"},
                        "sms_active":{"to":"{sms_to}","txt":"Home Internet is down!"},
                        "sms_normal":{"to":"{sms_to}","txt":"Home Internet is restored!"}},
                to_do = "app"),
        notifications = {
                "mail_internet_lost":{"subject":"{app_txt}"},
                "mail_internet_ok":{"subject":"{app_txt}"},
                "mail_network":{"subject":"Network Report - Lost={app_txt}","to":"{prime}"},
                "mail_ping_lost":{"subject":"{app_txt}"},
                "mail_ping_ok":{"subject":"{app_txt}"}},
        ntp_server = "192.168.15.1",
        power_ok = Input(
                duration = 2,
                effect_virtuals = {
                        "power_flag":Virtual(duration = 2,to_do = "do_me_up")},
                notifications = {
                        "cal_active":{"txt":"Home Power lost, electricity is down!"},
                        "cal_normal":{"txt":"Home Power restored, electricity is back!"},
                        "mail_active":{"subject":"The House is without electricity","to":"{prime}"},
                        "mail_normal":{"subject":"The House is back with electricity","to":"{prime}"},
                        "say_active":{"txt":"{tts_start} the house is without electricity{tts_end}"},
                        "say_normal":{"txt":"{tts_start} electrical power in the house is restored{tts_end}"},
                        "sms_active":{"to":"{sms_to}","txt":"Home Power lost, electricity is down!"},
                        "sms_normal":{"to":"{sms_to}","txt":"Home Power restored, electricity is back!"}},
                pin = "unipi:PI-Data,input,2"),
        role_me = "PI-Data"),
       ],

drivers = [
    Google_driver(
        application_name = "HomeEvents",
        client_secret_file = "client_secret.json",
        notifications = {
                "mail_cal_log":{"subject":"Google Calendar Notifier Generated Events - {app_txt} new items"},
                "mail_gmail_events":{"subject":"Google Calendar Events Report","to":"{prime}"}},
        scopes = "https://www.googleapis.com/auth/calendar"),
    Vera_driver(
        alexa_cmds = {"Curtains":{"off":"vera_open_curtains","on":"vera_close_curtains"}},
        notifications = {"mail_vera_parsing":{"subject":"Vera Parsing","to":"{prime}"}},
        vera_cmds = {
                "vera_clim_pars":Vera_cmd(log_nty = "Climatisation parameters received",vera_pars = ["82"],vera_type =
                "vera_close_curtains":Vera_cmd(log_nty = "Vera closed curtains",vera_pars = ["48"],vera_type = "RunSce
                "vera_house_mode":Vera_cmd(log_nty = "Vera House Mode Set to {home_occ}",vera_pars = [""],vera_type =
                "vera_is_holiday":Vera_cmd(log_nty = "Holiday parameter received",vera_pars = ["78"],vera_type = "RunS
                "vera_open_curtains":Vera_cmd(log_nty = "Vera opened curtains",vera_pars = ["49"],vera_type = "RunScer
                "vera_security_pars":Vera_cmd(log_nty = "Security parameters received",vera_pars = ["107"],vera_type =
        vera_port = 3480),
    Btle_driver(
        btle_blackout = Virtual(to_do = "app"),
        btle_gw_entry = ["PI-Garden","PI-Gate"],
        btle_gw_exit = ["PI-Security"],
        notifications = {
                "cal_b_entry_{id}":{"txt":"{app_txt}"},
                "cal_b_exit_{id}":{"txt":"{app_txt}"},
                "cal_b_refused_{id}":{"txt":"{app_txt}"},
                "ifttt_b_entry_{id}":{"txt":"entry"},
```

```
                "ifttt_b_exit_{id}":{"txt":"exit"},
                "ifttt_b_refused_{id}":{"txt":"refused_entry"},
                "mail_b_entry_{id}":{"subject":"{app_txt}","to":"{prime}"},
                "mail_b_exit_{id}":{"subject":"{app_txt}","to":"{prime}"},
                "mail_b_refused_{id}":{"subject":"{app_txt}","to":"{prime}"},
                "say_b_entry_{id}":{"txt":"{tts_start} {id} entered the house{tts_end}"},
                "say_b_exit_{id}":{"txt":"{tts_start} {id} left the house{tts_end}"},
                "say_b_refused_{id}":{"txt":"{tts_start} {id} is refused access to the house{tts_end}"}},
        role_me = "PI-Security"),
    ],

property=House(
            places = {"kitchen":Room(
                    contents = [
                        Cameras(items = {"cam_kitchen":Camera(cam_tpe = "foscam1", file_ID = "KC", ip = "192.168.15.40
                        Doors(items = {"door_kitchen_terras":Door(pin = ["ow:PI-Climate,12742ECC000000B4,DS2406,A,-15'
                        Windows(items = {
                                    "win_kitchen_left":Window(pin = ["ow:PI-Climate,12212ECC00000071,DS2406,A,-14'
                                    "win_kitchen_middle":Window(pin = ["ow:PI-Climate,12212ECC00000071,DS2406,B,-1
                                    "win_kitchen_right":Window(pin = ["ow:PI-Climate,12742ECC000000B4,DS2406,B,-15
                        Climate(
                                clim_makers = {
                                    "r23_front":Clim_SW(
                                            i_make = "warm",
                                            member_of = ["pump"],
                                            method_things = {"°C_fluid":Sensor(i_read = "°C", pin = "ow:PI-Climate
                                            pin = "unipi:PI-Climate,relay,5"),
                                    "r24_rear":Clim_SW(
                                            i_make = "warm",
                                            member_of = ["pump"],
                                            method_things = {"°C_fluid":Sensor(i_read = "°C", pin = "ow:PI-Climate
                                            pin = "unipi:PI-Climate,relay,10")},
                                clim_sensors = [Sensor(i_read = "°C",pin = "ow:PI-Climate,28503AAF040000A4,DS18B20,,-4
                                clim_targets = {
                                    "warm_sp":{
                                            "away":17.0,
                                            "comfort":1.0,
                                            "day":{
                                                    "off":10.0,
                                                    "on":{"00:00":17.0, "07:00":19.5, "10:00":18.5, "11:30":19.5,
                                                          "14:00":18.5, "17:30":19.5, "20:30":17.0}},
                                            "economy":-1.5,
                                            "sleep":17.0}},
                                my_assistant = True,
                                room_virtuals = {
                                    "{room}^clim_on":Virtual(
                                            copy_things = {
                                                    "twin_copy":Output(pin = "zw:{tc},buttonset,146,Status8")},to_
                                    "{room}^clim_pref":Virtual_R(
                                            copy_things = {
                                                    "twin_copy@-1":Output(pin = "zw:{tc},buttonset,171,Status8"),
                                                    "twin_copy@1":Output(pin = "zw:{tc},buttonset,171,Status4")},
                                            description_r = "Economy,Standard,Comfort",
                                            digital_range = "-1,0,1",
                                            to_do = "app")}),
                        Security(zone = "house"),
                        Sonos(items = [Sonos(ip = "192.168.15.111", sonos_type = "PLAY:1"),
                                       Sonos(ip = "192.168.15.83",sonos_type = "PLAY:1")]),
                        Ip_ping(items = {"echo_{room}":Ping(ip="192.168.15.157", ip_action = -1, spec_func = "Echo")}
                    room = "{room}"),
            }
)
```

The total configuration can become long, several thousand lines long, but this kind of detail one your home or apartment is exactly what is needed to make it work for your wishes.

Now that you have a feel for how it looks, let's zoom into the Entities section, followed with a section on the core, system and functional apps, the drivers and then the things and base things which are the basis of all things which closes the list.

Then we explain some concepts and then close with app briefings to get an idea of what complete functionality these apps offer.

We hope you enjoy discovering universal home control - Lucy

Entities

Entity	Description
Apartment	entity object - Apartment
Building	entity object - Building
Business	entity object - Business
House	entity object - House
Site	entity object - Site

APPS and Drivers

Core Apps

what_is	Description
copy_things	additional settings about a thing, adding features and logic
effect_virtuals	effects things can have on each others such as striking a value when another thing gets active or inactive
errors_warnings	A long list of messages, errors and warnings exist
method_things	additional settings about a thing, adding features and logic
notifications	Notifications are messages triggered by an app or a thing and they are delivered through different channels
path	referencing the physical hardware part for input things or output things
things	a thing is a physical device with connectivity which enables these objects to connect and exchange data
value_app	Logic via an app to determine the payload based on the app parameters
value_logic	Logic to automatically determine the payload based on time or other things
virtual	Virtuals are imaginary or abstract Things mostly originating from Apps but with real connectivity and data exchange and they make your configuration smart and easy to understand

System Apps

These Apps are main internal components of Lucy, they are the backbone of the harmonious whole user experience and setup

App_system	Description
Deployer	Manages and protects the things-controllers if multiple are deployed at a site and ensures automatic program version distribution and health check
Feeder	This App processes incoming commands from various sources and checks eligibility. Sources for the feeder could be Vera, TCP, UDP, IFTTT, Calendar, email, SMS, twitter,... Also included is the master and other things_controllers who need a service. The feeder generates a report with all the possible commands with parameters and if and who can execute such command.

App_system	Description
Forensics	Collects and reports cumulative data of all Things for a 24 hours period and assigns to every things_controller access to updated status reports linked to its role
Home_settings	App to define parameters such as home occupancy, is_holiday that are crucial aspects for most other functional apps
My_assistant	Advanced voice control with Amazon's Alexa, ok_Google, Siri Notes and for instructions from calendar and mail and for the Apps that are enabled to receive voice instructions
Network_controller	Protects the ip network. Scans all devices for ping latency, a summary is emailed at midnight
Notifier	Is the App to deliver all outgoing notifications to displays, buzzers, voice output or any other notification channel defined and available through a driver
Prj_parser	This is the home configuration compiler and syntax checker, it generates the binary objects that the apps will work from
Scenes_app	TBI/Scenes App, containing scenes that can be triggered in a calendar, by sms, by email, by twitter, by IFTTT, by Vera, by UDP, by TCP. The scene id's are the key for the feeder to know what to do when the event_id is received through one of the feeder's channels
Site_settings	Defines the geographical location (needed for sunset and sunrise), the email names used and some site specific parameters such as currency and temperature scale
Site_tasker	the app that allows certain roles to assume a multi site role
Things_additions	Some Apps have something to do with specific things (such as a sirens_test with an Alarm_siren) or you need to define an App that collects data for every thing defined, then this App is the one to use
Things_sync	The process of ensuring that the as_is and to_be state of a Thing is there where it needs to be for the system to work as a whole

Driver Apps

These Apps interface to gateways, bridges of other suppliers or make a connection to external services and Apps:

Driver	Description
Btle_driver	Bluetooth low energy beacons! They are used to provide access and registration to the house for cars and motor's or for other mobile services (heart monitoring..)
Camera_driver	Driver for interfacing with camera's for pictures (pic), video (vid) and date/time (datim) setting. Per cameratype the url's should be defined
Daikin_driver	This driver enables the daikin room airconditioners to be used in the climatization app as room coolers, heaters, ventilators or air dryers
Dropbox_driver	This driver defines the dropbox handler and the dropbox paths to the folders where logs, cumulative data and program scripts is to be updated. This updating occurs automatically at the end of the day before log files or cumulative files are refreshed for a new day
Google_driver	This is the google calendar driver to read the calendar and read person and device credentials (f.i. is entrance allowed?) and to execute events such as holidays, heating up guest rooms, triggering irrigation or lights for special occasions
HAP_driver	is the driver for HAP Home Apple Protocol
Hue_driver	This is the Philips Hue driver, to interface to the Philips Hue bridge that drives all the Hue devices
Ifttt_driver	IFTTT IF THIS THEN THAT driver, both as a trigger with 'maker' and as an google assistant applet processor, These allow ok Google to work, see instructions and the trigger url and reverse web hook url

Driver	Description
Ikea_driver	This is the IKEA Tradfri driver, to interface to the IKEA Tradfri light gateways that drives all the Tradfri devices
Ip_building_driver	is the driver for the IP-Building modules
KNX_driver	is the driver for KNX
Log_driver	This is the logging that the apps will use to keep a trace of what is happening
Loxone_driver	is the driver for the loxone master module ip interface
Lutron_driver	is the driver for Lutron
Modbus_driver	is the driver for Modbus
Netatmo_wlc_driver	Netatmo welcome system driver -> make a different id compared with the netatmo weather station. Netatmo Welcome: name your netatmo welcome camera and tags.
Netatmo_ws_driver	Netatmo weather station driver -> make a different id compared with the netatmo welcome system. Name your inhouse and outhouse station to the room where they are placed, e.g. 'garden','office' and rainmeter / windmeter for these modules
Niko_driver	is the driver for Niko
One_wire_driver	Handles the 1 wire sensors, 1 wire devices and iButtons
Piface_driver	Reads and writes binary inputs and outputs on the piface hat
Slack_driver	is the driver for the Slack interface
Sms_driver	GSM SMS driver, both as a notification sender of messages as a receiver of remote commands, via the hologram nova modem, see http://hologram.io
Somfy_driver	is the driver for Somfy
Sonos_driver	The Sonos Driver to play notifications through a sonos setup. Define the sonos speakers with the Sonos() thing!
Tcp_driver	TCP Driver
Text2speech_driver	contains the parameters for the text to speech parameters and credentials, for Microsoft speech services, for a free subscription to speech services, 5000 messages per month goto https://account.windowsazure.com
Twitter_driver	is the driver for Twitter
Udp_driver	UDP Driver
Unipi_driver	Reads and writes binary and analog inputs and outputs on the unipi and neutron hats using evok, see http://www.unipi.technology
Usb_driver	Driver for interfacing with usb serial devices.
Vera_driver	The driver to vera devices used for zwave communications and the scene's used for voice triggering. Vera has the most complete and open zwave and zigbee gateway on the market, supporting the most diverse zwave things

Functional Apps

These Apps deliver the functional richness of Lucy and can be expanded in the future to the most complete integrated solution available anywhere. The beauty lies in the integration of the different functions to deliver a harmonious user experience.

App_functional	Description
Access_keys	is a dictionary of access keys for persons with access keys or vehicles with btle tags that define the access rights granted by the controller where the access key is registered

App_functional	Description
Access_manager	Defines access for people or items such as cars. For people access_point's register incoming and outgoing persons and grant access given proper credentials. For items, bluetooth low energy keys can be used. This App defines the access keys and what they do, and the respective notifications and ifttt hooks
Climate_manager	The Climatisation App manages all the multi room climate systems and individual climate makers working solo for a room
Irrigation_manager	The Irrigation App, whereby based on past and furture rain precipitation plant watering is optimized
Light_manager	Defines the things_controller managing the light process and steering the Hue bridges, the IKEA tradfri gateways and any Vera light devices
Security_manager	This App defines all the security settings and notifications that all security systems inherit
Utilities	Dictionary of utilities with a designated thingscontroller to manage
Utility	An utility description with meters, sensors, costs, scenes
Utility_storage	An utility storage such as a battery or a water tank
Wincover_manager	Manages and contain options for the defined Door and Window coverings

Apps for Rooms or Places

These Apps can be defined in a place or a room and decorate it with functionality and methods.

App_place	Description
Access_ways	Define a list or a dict of one or multiple instances of Access_point or Access_trigger
Cameras	Defining IP camera's and camera groups. Pictures will be taken at specified events through notifications and the camera group memberships
Climate	Defines the valves, sensors, climate devices such as heaters, coolers, ventilation and humidification regulators and there settings in sleep, away, holiday or day mode
Climate_system	Climate Systems are typically multi room systems that drive multiple makers in multiple rooms with a central energy source such as a boiler and a pump to drive the energy around. This opposed to solo makers, which impact a climate just in one room and do not have not a multi system to back them. This App controls boilers, pumps, hvac devices in a classical radiator or underfloor heating (or combined) and everything else to give you excellent climate comfort given the context of comfort, standard or economy, the rooms that are in scope and the home occupancy, normal day, away, sleeping or on holiday
Control	Generic Motor and Output setting such as swimpool pumps, wine cellar temperatures and any other things to control
Doorbell	Most doorbells drive a bell button signal and have a way to signal agreement to open or lock one or more doors. You can also declare a doorbell inside a door.
Doors	A door? Is it not a simple input that states if the door is closed or not? A door can have a raft of method_things including all types of notifications, lights or triggers to open or close the door. To name a few method_things; optical beams to open or close the door, light to open, to close, a light when opening or closing or a night light. Or a timer on the open or close duration that triggers something. Or to define which area the door/window gives access to which is used in the security App
Ip_ping	Register network devices with special parameters for non pingable items and 'special' ip addresses. Devices defined with their ip address such as sonos, camera's and things_controllers are automatically added to the ip_ping list

App_place	Description
Irrigation_points	Irrigation point for water delivery to plants, with the normalized duration irrigation time in minutes
Irrigation_system	Interface to the garden irrigation pump(s), activation/deactivation buttons and the watering valves
Lights	All types of light, dimmable and binary with a special option to have toggle lights with a light sensor
Mailbox_alert	To notify incoming and outgoing regular post mail through a magnetic or optical switch in the mailbox
Monitor	Device monitoring such as waste water levels, wine cellar temperatures and any other things data to watch and to notify if set boundaries are crossed
Music_players	Any of the music_players supported such as sonos
Phone_dialer	This is the driver to an old fashioned phone dialer to deliver preset voice messages in case of fire, alarm or loss of internet or power
Pool_manager	This is the app for swimpool management
Security	This App is defined per room and contains the Fire/Alarm detectors and sirens and the zones linked to the place or room
Security_system	Advanced interface and definition of a security system, either stand alone or integrated as a master or slave subsystem. The role drives the alarm and fire detectors and the whole security logic
Things_controllers	This structure defines a Things Controller such as 'Raspberry','Ubuntu','MAC_OS','Arduino','Vera','Hue','Ikea','Eds','Unipi_Evok'
Weather_station	wunderground and darksky is used for the weather forecast, Netatmo weather station is used for the wind, rain and other semi real time data. The role involves the weather forecast, weather station (wind, rain, sunshine) and irrigation based on forecast. Several direct input sensors are supported for rain, wind, temperature, humidity and soil moisture
Windows	A door? Is it not a simple input that states if the door is closed or not? A door can have a raft of method_things including all types of notifications, lights or triggers to open or close the door. To name a few method_things; optical beams to open or close the door, light to open, to close, a light when opening or closing or a night light. Or a timer on the open or close duration that triggers something. Or to define which area the door/window gives access to which is used in the security App

Basis of Things

Thing_base	Description
Dimmer	Dimmer description, works also for the derived classes such as Motor, Dim_light and Color_light
Input	Input description, works also for the derived classes
Output	Output description, works also for the derived classes
Sensor	Any temperature sensor
Str_device	Str_device, a device containing text
Virtual	Virtual's are an important vehicle to obtain smart configurations whereby interaction and dependencies between Things and Apps can be arranged, think of Virtuals as abstract rather than real things
Virtual_A	Analog type Virtual
Virtual_R	Range Virtual's do not just have a binary state such as ordinary virtuals but they can have a value in a range of consecutive integer numbers

The Things

Thing_sub	Description
Access	Access description, works also for the derived classes
Access_point	Access points are devices that read iButtons, smart cards, finger print scanners, facial recognition devices and emit an access key string to the security guardian that is interpreted for access rights and consequential access scene execution. Access controllers follow the propriety data exchange protocol with the security things_controller and have a direction: enter, exit or select
Access_trigger	Access trigger are inputs and when activated, they submit an access_scene id to the security guardian for scene processing
Alarm_detector	Alarm_detector description
Alarm_siren	Alarm_siren description
Ble_gw_th	is an internal generated thing for every beacon for every gateway to hold the rssi value or zero when lost
Button	Input description, works also for the derived classes
Camera	is a network function to check the presence of a thing through a ping
Clim_ANY	Climate Anything, clim_ANY is a climate device that has a string setting and value can be set with value_logic. For example fan direction could be something to define and hold here in combination with value_logic is this climate maker multi purpose friend
Clim_DM	Climate Dimmer, clim_DM is a climate device that has a 0% to 100% setting and value can be set with value_logic. Typically this is for 3 way or motorised valves that can be more opened automatically with the outside temperature as more hot water can be channeled through instead of being looped back. Other purposes are in combination with i_make='wind','vent','fan' and in combination with value_logic is this climate maker multi purpose friend
Clim_energy_DM	Climate energy Dimmer, is a device that has a 0% to 100% setting and value can be set with value_logic. Typically this is for heat/cool pump generation. Specify i_make.
Clim_energy_SW	Climate energy Switch, is a binary on/off switch for gas or electricity heaters or coolers. Specify i_make.
Clim_SP	Climate Setpoint, Clim_SP is a climate device that has a (temperature) set-point setting, it is therefore an analog device
Clim_SW	Climate Switch, Clim_SW (switch) is a binary on/off climate switch such as an on/off valve. you can add a temperature sensor which measures the passing through cooling or heating liquids
Color_light	Color_light description, works also for the derived classes
Dim_light	Dim_light description, works also for the derived classes
Door	Door and Window methods are the same
Fake_meter	Virtual meter for the registration of calculated meter values, for example the water consumption in a T pipe when real meters work for 2 of the 3 legs.
Fake_sensor	Virtual sensor for the registration of calculated sensor values, for example the electricity power taken to make the sum of power users zero.
Fire_detector	Input description, works also for the derived classes
Irr	Irr description, works also for the derived classes
Light	Light description, works also for the derived classes
Meter	Meter to register utility usage or flow
Motor	Motor description, works also for the derived classes
Optical	Input description, works also for the derived classes

Thing_sub	Description
Ping	is a network function to check the presence of a thing through a ping
Rain_gauge	Rain gauge meter device
Sensor_switch	An Input switch which is activated by something such as a high temperature like a thermostat
Sonos	is a network function to check the presence of a thing through a ping
Switch	Input description, works also for the derived classes
Win_cover	Window covering
Wind_gust	Wind gust device
Wind_speed	Wind speed meter device
Window	Door and Window methods are the same

Thingscontrollers

Things_controller	Description
Arduino	controller of things - Arduino
Daikin	controller of things - Daikin
Eds	controller of things - EDS
HAP	controller of things - HAP
Hue	controller of things - Hue
Ikea	controller of things - Ikea
IP_Building	controller of things - IP_Building
KNX	controller of things - KNX
Loxone	controller of things - Loxone
Lutron	controller of things - Lutron
MAC_OS	controller of things - MAC_OS
Modbus	controller of things - Modbus
Niko	controller of things - Niko
Raspi	controller of things - Raspi
Renson	Renson Healthbox controller
Somfy	controller of things - Somfy
Ubuntu	controller of things - Ubuntu
Unipi_Evok	controller of things - Unipi_Evok
Vera	controller of things - Vera

Section "What is?"

In the next sections, a few design concepts are explained, which helps to understand the uniqueness and specifics of this Universal Home Control solution.

Virtual Things : make your configuration smart

When pondering the strategy to separate home specific configuration from programming logic, the idea emerged of using virtual things to make the configuration smart and to have a powerful design mechanism to have logic in the home configuration file. It worked out better than anticipated, and the result is a very powerful combination. The negatives, cluttering the configuration file with virtuals from apps or own invented 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 separating home configuration data from the processing code.

Let's see how this works and we will cover the different virtuals, things that can have an imaginary payload not linked to the real world as real things can:

- app virtuals : they are created and used in the context of an App. Some are mandatory and have to be defined in order for the App to work.
- effect virtuals : things that can do a play: receive an effect or splash an effect on a parent

To illustrate the syntax in the example below, the Network_controller App has 1 "App" virtual 'internet_lost'. This is an object starting with **Virtual()**, used for signaling the loss of internet in other places in the PHCF.

Another virtual 'power_flag', further down in the example, is a (non app) effect_virtual where the play is an effect: when power_ok becomes active, self (power_flag) is made active.

from project.py tree:(o:Network_controller)

```
# --> project.py :<dk:project,o:Project,kw:apps,lp:0,o:Network_controller>

from lucy_app import *

Network_controller(
    IP_WAN = "127.0.0.1",
    gateway = "192.168.15.1",
    internet_lost = Virtual(
            notifications = {
                    "active":[
                        Say(txt='{tts_start} the internet connection is down{tts_end}', ceiling=None, times=1, overric
                        Sms(to='{prime}', txt='Home Internet is down!', override=None, ceiling=None)],
                    "inactive":[
                        Say(txt='{tts_start} the internet connection returned{tts_end}', ceiling=None, times=1, overri
                        Sms(to='{prime}', txt='Home Internet is restored!', override=None, ceiling=None)]}),
    internet_ping_name = "internet_access",
    internet_ping_repeat = 15,
    notifications = {
            "internet_lost":Mail(subject='{app_txt}', to=None, cams=None, cam_groups=None, passes=0, body_file='', fil
            "internet_ok":Mail(subject='{app_txt}', to=None, cams=None, cam_groups=None, passes=0, body_file='', files
            "network":Mail(subject='Network Report - Lost={app_txt}', to='{prime}', cams=None, cam_groups=None, passes
    ntp_server = "192.168.15.1",
    power_ok = Input(
            active = 0,
            duration = 2,
            effect_virtuals = {
                    "power_flag":Virtual(
                            duration = 2,
                            play = Effect(maker='parent', condition='become_inactive', effect='make_active', taker='se
            notifications = {
                    "active":[
                        Mail(subject='The House is back with electricity', to='{prime}', cams=None, cam_groups=None, p
                        Say(txt='{tts_start} electrical power in the house is restored{tts_end}', ceiling=None, times=
                        Cal(txt='Home Power restored, electricity is back!', summary='', ceiling=None),
                        Sms(to='{prime}', txt='Home Power restored, electricity is back!', override=None, ceiling=None
                    "inactive":[
                        Mail(subject='The House is without electricity', to='{prime}', cams=None, cam_groups=None, pas
                        Say(txt='{tts_start} the house is without electricity{tts_end}', ceiling=None, times=1, overri
                        Cal(txt='Home Power lost, electricity is down!', summary='', ceiling=None),
                        Sms(to='{prime}', txt='Home Power lost, electricity is down!', override=None, ceiling=None)]},
            path = "unipi:PI-Stats,input,3"),
    role_me = "PI-Stats")
```

The security_system app has more than a dozen app virtuals defined, to be used in your PHDF.

As you learned in the example, it is possible to define own virtual things simply by creating them where needed with a name and an effect that defines the purpose. These are called effect_virtuals,
The config parser "knows" when a virtual is referenced when it has its name registered in a previous context. Virtuals can do something on the thing(s) they are linked to (the parent acts) or inverse (self acts on the parent) and this is called a play.

The play is a named tuple "Effect" with fields: ["maker", "condition", "effect", "taker", "duration", "delay"].

- maker = self, parent : the thing that takes the initiative, the thing that defines the effect_virtual (parent) or the effect_virtual itself (self)
- condition = what condition activates the effect to make
- effect = what to do on the taker
- taker = the thing that the maker to act upon (must be the opposite of maker)
- duration = (optional) seconds for the duration of the effect
- delay = (optional) seconds before the effect to happen

For a Virtual Range thing, the 'condition' can be a list of discrete values (digital_range) or a from - till range for an analog range virtual

See below the valid values for each of the fields:

from app_obj.conf:

```
[VIRTUAL_PARAMETERS]

play={"maker":    ["self","parent"],
        "condition":["become_active","become_inactive","i_change","when_active","when_inactive"],
        "effect":   ["make_active","make_inactive","make_toggle","make_same", "active_freeze","inactive_freeze","make_
        "taker":    ["parent","self"],
        "delay":    None,
        "duration": None}
```

as an example of frequently used virtual to_do's (many more are available) when a Virtual is linked with another Thing:

For effect_virtuals, the syntax of the play is built to almost have a readable sentence, just read the examples below:

Examples of plays where the parent acts on the Virtual self:

```
play=Effect(maker="parent", condition="become_active",   effect="make_active",   taker="self")
play=Effect(maker="parent", condition="become_inactive", effect="make_inactive", taker="self")
play=Effect(maker="parent", condition="changes",         effect="make_toggle",   taker="self")
play=Effect(maker="parent", condition="changes",         effect="make_same",     taker="self")
play=Effect(maker="parent", condition="become_active",   effect="make_inactive", taker="self")
```

Examples of plays where the self Virtual acts on the parent:

```
play=Effect(maker="self", condition="become_active",   effect="make_active",     taker="parent")
play=Effect(maker="self", condition="become_inactive", effect="make_active",     taker="parent")
play=Effect(maker="self", condition="become_inactive", effect="make_inactive",   taker="parent")
play=Effect(maker="self", condition="become_active",   effect="make_inactive",   taker="parent")
play=Effect(maker="self", condition="changes",         effect="make_toggle",     taker="parent")
play=Effect(maker="self", condition="when_inactive",   effect="freeze_inactive", taker="parent")
```

Example of a Virtual Range play:

For a Virtual Range thing, the 'condition' can be a list of discrete values (digital_range) or a from - till range for an analog range virtual

```
play=Effect(maker="self", condition=[0,2], effect="freeze_inactive", taker="parent")
```

Further reading on how to use virtuals.

Virtuals and Things are linked through properties and this brings us nicely to the next section.

Properties : adding features and logic

Properties are extra features for a thing and they come in the form of method_things, copy_things, notifications and the making of a payload with effect_virtuals and value_logic.

Some or all fields are mandatory or can be repetitive in a list or dictionary form. The fields of this dictionary are checked for eligibility and it can be best showed through some examples.

Properties in apps are attributes such as the temperature scale °C or °F in site_settings

Properties are also available in the definition of Inputs, Outputs, Lights, Buttons, Dimmers,.. to define a dictionary of additional attributes.

method properties : method_things

These are properties associated to the specific nature of a thing, such as a door which sometimes have a pulse contact to open or close the door.

There a keyword 'method_things=' is to be added followed by a dictionary structure.

Several types of properties exist and can be activated :

1. Inherited properties : properties of basic things they pass on to their offspring
2. Circumstantial properties : such as the ones for a door or window which are specific for the thing or the app

Some method properties have a special meaning, such as a toggle_button in an output thing (it will toggle the output when the input becomes active) and in the same way it is possible to define virtuals in properties to activate things.

copy properties : copy_things

Copy properties add inter thing logic and make your configuration smartDefining or referring to a Virtual.

One thing can influence another, as for a Carbon copy thing, or twin copy when both things influence each other when they change value.

Other things can act as trigger or can be triggered.

notification properties

Notification properties : for every context there is a list of possible notifications and you are free to configure these to your needs

Such as in the example below where you want a voice notification if the internet becomes unavailable or reappears.

Properties are checked for eligibility during parsing of site.conf and error messages are generated when issues are noticed.

Most errors stop the program altogether, as there is zero tolerance to parsing errors to ensure proper operations.

Some examples will demonstrate how properties work.

Example of a notification property

Use notification properties to issue a voice message and an sms when internet access or electrical power is lost or restored or when a network ping on a device fails or works again.

These properties are simply notifications:

from project.py tree:(o:Network_controller)

```
## --> project.py :<dk:project,o:Project,kw:apps,lp:0,o:Network_controller>

from lucy_app import *
```

```
Network_controller(
    IP_WAN = "127.0.0.1",
    gateway = "192.168.15.1",
    internet_lost = Virtual(
            notifications = {
                    "active":[
                        Say(txt='{tts_start} the internet connection is down{tts_end}', ceiling=None, times=1, overri
                        Sms(to='{prime}', txt='Home Internet is down!', override=None, ceiling=None)],
                    "inactive":[
                        Say(txt='{tts_start} the internet connection returned{tts_end}', ceiling=None, times=1, overr
                        Sms(to='{prime}', txt='Home Internet is restored!', override=None, ceiling=None)]}),
    internet_ping_name = "internet_access",
    internet_ping_repeat = 15,
    notifications = {
            "internet_lost":Mail(subject='{app_txt}', to=None, cams=None, cam_groups=None, passes=0, body_file='', fil
            "internet_ok":Mail(subject='{app_txt}', to=None, cams=None, cam_groups=None, passes=0, body_file='', files
            "network":Mail(subject='Network Report - Lost={app_txt}', to='{prime}', cams=None, cam_groups=None, passes
    ntp_server = "192.168.15.1",
    power_ok = Input(
            active = 0,
            duration = 2,
            effect_virtuals = {
                    "power_flag":Virtual(
                            duration = 2,
                            play = Effect(maker='parent', condition='become_inactive', effect='make_active', taker='se
            notifications = {
                    "active":[
                        Mail(subject='The House is back with electricity', to='{prime}', cams=None, cam_groups=None, p
                        Say(txt='{tts_start} electrical power in the house is restored{tts_end}', ceiling=None, times=
                        Cal(txt='Home Power restored, electricity is back!', summary='', ceiling=None),
                        Sms(to='{prime}', txt='Home Power restored, electricity is back!', override=None, ceiling=None
                    "inactive":[
                        Mail(subject='The House is without electricity', to='{prime}', cams=None, cam_groups=None, pas
                        Say(txt='{tts_start} the house is without electricity{tts_end}', ceiling=None, times=1, overri
                        Cal(txt='Home Power lost, electricity is down!', summary='', ceiling=None),
                        Sms(to='{prime}', txt='Home Power lost, electricity is down!', override=None, ceiling=None)]},
            path = "unipi:PI-Stats,input,3"),
    role_me = "PI-Stats")
```

Many notification properties exist and some are available through inheritance from things, and others are circumstantial, such as the Weather_station ones.

Using Virtual's in copy properties

A more complex example involves the use of virtuals for post mail handling, see below an example : Mailbox_alert.

Through the definition of the properties, a voice and an email message are generated when new mail arrives and an email message is sent when the mail is removed. The email message contains camera shots from the specified camera group which will loop 2 times (2 shots per camera 4 seconds in between).

More importantly, a self invented effect_virtual "have_mail" is maintained and set high when there is new mail and set low when the mail is removed. This virtual flag "have_mail" is then available somewhere else to activate a light or act on other things or apps.

from project.py tree:(o:Mailbox_alert)

```
## --> project.py :<dk:project,o:Project,kw:property,lp:0,o:House,kw:places,dk:street,o:Place,kw:contents,lp:5,o:Mailb

from lucy_app import *

Mailbox_alert(mail_in = Input(
            effect_virtuals = {
                    "have_mail":Virtual(
                            play = Effect(maker='parent', condition='become_active', effect='make_active', taker='self
            notifications = {
                    "active":[
                        Mail(subject='{thing+is}', to='{prime}', cams=None, cam_groups=['cams_gate'], passes=2, body_f
                        Say(txt='{tts_start} new post arrived by mail{tts_end}', ceiling=None, times=1, override=None,
```

```
                Cal(txt='Post delivered in the mailbox', summary='', ceiling=None)]},
        path = "unipi:PI-Garden,input,6"),mail_out = Input(
        effect_virtuals = {
                "have_mail":Virtual(
                        play = Effect(maker='parent', condition='become_inactive', effect='make_inactive', taker='
        notifications = {
                "active":Mail(subject='{thing+is}', to=None, cams=None, cam_groups=['cams_gate'], passes=1, body_t
        path = "unipi:PI-Garden,input,7"),role_me = "PI-Garden")
```

Further reading on properties.

Value_Logic : set payload based on conditions, time events or app scripts

Home automation results in things receiving a payload (typical jargon) based on very complex circumstances and value_logic is a reasonable simple solution to capture this complexity in an understandable manner.

Imagine the payload is determined on time conditions or status of other things (assign), or the current payload is frozen (freeze) until this lock is lifted. Additionally one could have conditions to keep the thing into a inactive state (disable) or just have conditions that must be satisfied to allow alteration of the thing (enable). And when the payload is forced over, as in the example of light dimming where you desire the intensity to be different from the one determined by value_logic, then you lock your preference. When this temporary payload lock is lifted is also automated.

So value_logic is the process of assigning a value to a thing and 4 logical possibilities exist: - assign : assign a payload based on a time occurrence or one of the special influencer things, real or virtuals - freeze : conditions in which payload is frozen and cannot not change - enable : all conditions to satisfy before an assign payload can apply - disable: any of the conditions True will force the payload to inactive (binary type things) or zero (analog or range type things)

So value logic is a dictionary with one or more of these 4 conditions (assign, freeze, enable, disable) with payload influencing properties that is available for output derived things. The impact of each of these conditions can be delayed a number of seconds before or after they apply.

Value_logic can generate values for more complex circumstances and it that case a value_app is the solution whereby a program script is called in a controlled environment to obtain a value.

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

2. in this example curtains open or close 15 minutes after sunrise or after 7:30 what comes last and before sunset or 5pm whatever comes first and not the rest of the day.

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

3. this example sets the ventilation speed (0..100) based on the occupancy modes of the house if it is day or night

```
"ventilation_speed": Dimmer(path="pi:PI-Climate,1",
        value_logic={"assign":{"is_holiday":"10","away":"10","sleep":"50","is_day":"100"})
```

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(path="pi:AR-Hall,1",
        value_logic={"assign":{"sleep":"10","away":"0","is_holiday":"0","00:00":"25","sunset":"100"}})
```

5. for philips hue things, 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..100) till noon, then tropical_twilight at 75 and finally from sunset savanna_sunset at 50.

```
"office_ledstrip": Dimmer(path="hue:Hue_Bridge,{:}",
        value_logic={
                "assign":{"is_room_secure":"0","00:00":"0","sunrise":"20,spring_blossom","12:00":"75,tropical_twi
        properties={"type":"RGB"})
```

6. the most complete example is for a window covering (a sun blocker screen). The screen is down over the window from one hour after sunrise till 2 hours before sunset if the sunshine is more than 75% and the temperature outside is above 22 degrees. But if the windspeed is more than 50 km/hour or it is raining, then the sunscreen is up (zero). To have latency there is a enable delay of 600 secs so that a sudden cloud do not flip flop the screens. But rain or wind will up the screens for at least 15 minutes after they disappear.

```
"sun_screen": Win_cover(path="vera:{:}",
        value_logic={
                "assign":{"sunrise+01:00":"100","sunset-02:00":"0"},
                "enable":{"sun_light_wc>75","°C_outdoor_wc>22"},"enable_delay":{"after":600},
                "disable":{"wind_speed_wc>50","raining_wc"},    "disable_delay":{"after":900}})
```

For more information, see Value_logic

Defining Things

Real Things need a reference to the physical world, and **path=** is the way to make this link and it is therefore mandatory. Path definitions come in different types and prj_parser checks that a Button cannot be defined as a temperature path, which can only be assigned to a sensor.

Path types define the physical input / output aspect and they have to follow a specific syntax:

path Type	Description	Syntax
piface	thing linked to a raspberry with piface	piface:{pi_name},{do,di},{hw_pin}
unipi	thing linked to a raspberry with a unipi or Evok webserver	unipi:{pi_name},{relay,input,ai,ao},{hw_pin}
arduino	thing linked to a arduino with things_sync protocol	arduino:{pi_name},{"di","do","ao"},{hw_pin}
ow	thing linked to a 1wire device	ow:{A or B},{1w_device} if it concerns a digital device else ow:{1w_device} for a sensor
zw	thing linked to a vera zwave device	zw:{zw_name},{zw_dev_nr},{zw_attr}
vera	vera scene or device	vera:{gw_name} or {:} -> given name
ikea	Ikea tradfri light element	ikea:{gw_name},{thing_name} or {:} -> given name
hue	Philips Hue light element	hue:{gw_name},{thing_name} or {:} -> given name
lutron	Lutron light element or switch	lutron:{gw_name},{thing_name} or {:} -> given name
daikin	Daikin soll element, default is soll temperature	daikin:{gw_name},{thing_name} or {:} -> given name
knx	KNX thing	knx:{gw_name},{thing_name} or {:} -> given name
modbus	Modbus thing	modbus:{gw_name},{thing_name} or {:} -> given name
niko	niko thing	niko:{gw_name},{thing_name} or {:} -> given name

path Type	Description	Syntax
somfy	somfy thing	somfy:{gw_name},{thing_name} or {:} -> given name
shelly	shelly thing	shelly:{ip},{channel} (channel is optional and only when white shelly mode is used)
tcp	a tcp message	tcp:{pi_name},{ip},{port},'ip_txt'
udp	a udp message	udp:{pi_name},{ip},{port},'ip_txt'
usb	a usb serial device such as an arduino	usb:{pi_name},{usb_driver setting_id}
na_tag	a netatmo movement tag	na_tag:{netatmo name of tag}

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.

Path definition Examples:

```
"iButton_out_red":     Light(path="pi:AR-1W-OUT,do,1",duration=2)        # path is an arduino, output path 1
"door_kitchen_terras": Door(path="ow:A,12742ECC000000B4")                # path is one wire DS2406 path A
"HVAC_test":           Clim_SP(path="zw:p_switch,135,1",i_make={"wind"}) # path is zw device 135 variable 1
"wall_light":          Light(path="vera:{:}")                            # path is vera device named wall_light
"guest_light":         Dim_light(path="ikea:Ikea_Tradfri,{:}")           # path is tradfri device guest_light
"bed_candle":          Dim_light(path="hue:Hue_Bridge2,{:}")             # path is on Hue Bridge 2, named bed candle
"temp_outside":        Sensor(path="°C:289574D906000075")                # path is 1w temp sensor
"tcp_light":           Light(path="tcp:PI-Light,192.168.15.76,5000,'{name}-{val}'") # path is a tcp message that will
"udp_light":           Light(path="udp:PI-Light,192.168.15.76,5000,'{val}'")        # path is a udp message that will
"iButton_out_garden":  Access_point(path="usb:PI-Garden,serial_arduino,string"}     # path is string data from a seria
```

Notifications

Notifications is the process of delivering messages through a Notification Channel when an app or a thing reaches a status causing a trigger. These triggers are proposed and documented in every app and every type of thing under the banner "Notifications".

All notifications are defined in the Home Configuration File as these are highly user specific. Various Notification Channels exists such as sending an email, saying something, sending an SMS, ...

For continuation when the internet is unavailable and to have no latency, all say messages are generated text to speech to wav files and stored ready to go on the local notifier controller.

Every notification is a named tuple. Currently the following notification channels exist, some of them are work in progress (TBI) and some new channels could be defined in the future (snapchat, rss feeds, etc..):

prefix	effect	comment
mail	email a message to the designated recipients	the subject text is updated with the tag text. Additional parameters apply such as camera groups to add picture shots to the email as well as body file, added files and subject formatting
log	log message	The text message will logged.
say	text to speech message as will voiced through the sound system	either in one room, or in all the premises with possibility to repeat a few times and no matter what (even when sleeping) for fire or alarm notifications
sms	gsm sms message	destination is a comma separated list of phone numbers. A phone must be an E.164 formatted string with a '+' sign, and the message string

prefix	effect	comment
udp or tcp	ip message	is ip address, ip port and message
zw	zwave call via vera	is the zwave device name, the device number and the variable value, see the zw pin definition
vera	vera control ltd vera call	one of the vera scene calls can be activated, see the vera pin definition
ifttt	IFTTT scene call	generates an ifttt maker scene (json) call, whereby the scene id is the nty name, value1 is the room or the option, value2 is the io_name and value3 is the io_state. **This allows complete integration between IFTTT and Lucy to allow an intensive personalisation of needs**
calendar	generate a calendar event with the text specified, duration=0, date & time of the notification	to log event in a calendar when a car left, or the home was armed/unarmed..
slack	generate a slack group message	TBI
buzzer	generate a buzzer sound for a certain duration and pitch	TBI
display	generate a display notification	
twitter	generate a tweet	TBI

For a notification example, please go back to the Properties subsection.

There is a whole raft of automatic generated tags that you can use in the email subject or txt section of your payload message. These tags make it easy to format the notification and it includes text generated by the app to provide understanding to the context of the notification.

see **notifier** for more details on the notification syntax.

Short App Briefings

For a few selected apps, a short briefing text is made to let you build an idea of the functionality currently available.

Access App

Access control is a very important aspect to any home automation system and significant effort and time has been devoted to design a solid future proof solution.

Access control could be implemented for a variety of physical access verification methods, being card readers, 1 wire buttons or more advanced fingerprint or facial recognition.

Therefore, the concept of access points is that all these possible verification methods result in an access key that is then forwarded to the security guardian to determine access to the scene linked to this access point for this key.

The system "knows" through the definition of the access point if a registration concerns an entry or an exit. Access points that have a selectable access direction will transmit with the access key also the access direction that has to be applied (entering or leaving).

Many access points can exist, to register the entrance of the property, or a safe room or any other room as each access_key / access_point combination results in a set of specific actions 'granting access'.

So when an identification is successful at the access point, then the access key is presented to the security controller.

Firstly, the access key will be verified against the authorization access key database and rejected if it not assigned to an access entity, assuming in the example that this is a person.

Assuming this test passes, the access_point, access_key combination will be tested against the access rights database and if positive, a access_event key is retrieved to lookup the associated actions.

A further restriction can be inserted for time bound access, whereby the person should be registered in the Google calendar before access can be considered.

This access_event_key points to a scene, which is typically a list of things (real and virtual) to be given a payload.

Everything is logged with detailed reporting and advanced notification possibilities.

This flexibility allows a very complete access solution that probably solves most home access issues and even allows access controlling any set of things, being door openers or anything else.

For mobile access control such as cars, motors or people, blue tooth low energy beacons can be used as access keys, see **Btle_driver** for more information.

Climate App

Climatisation is a crucial component of a home control system and significant time has been invested to make this rich in functionality and options.

Climatisation includes heating, cooling and proper ventilation and humidity control and the app is designed to cover these needs. Most home control solutions involve a nice thermostat and some valve control units. Problematic is the low reliability with battery powered wireless devices.

Although the app allows the use of such devices, a house is built for many decades and the continues maintenance of such devices is surely a major enthusiasm put off on the long turn and therefore we need to offer alternatives.

With the use of 1 wire temperature sensors and 24V valves, a reliable system can be built that works decades with little or no maintenance whatsoever as it should be.

In the same room or building you can combine multiple climate sources, such as a daikin multisplit in combination with a traditional boiler heating system. They are put to work seamlessly together for your climate needs. Even when you just have a classical multi room radiator system or underfloor heating or even a complex system of radiators and underfloor heaters and a hot water tank with motorized valves and manifolds.

The climatisation mode in effect at a given moment in the house can be cooling, none or heating, and this depends on the outside temperature. This switch from heating to cooling and the inverse happens automatically and is announced with the usual system of notifications.

A climatisation comfort setting exists and can be set for the whole house or just for a room or set of rooms and it can be "comfort","normal","economy". The difference between comfort and economy is a deviation from the target temperature (colder or warmer).

Rooms can be switched off when not in use and then a maintain temperature is enforced.

All temperature target settings are influenced by the occupancy mode, sleep, away or day mode, the time of the day and day of the week and if the house is set in holidays mode.

Every 5 seconds, the room temperatures are checked against the target and valves, pump and heaters/coolers are actualized. The pump speed depends on the number of heating/cooling devices needing energy and the heater temperature range depends on the outside temperature, colder outside means higher boiler temperatures. This ensures that energy is spent only there where needed and justly the amount to reach the programmed setpoint level.

The air removal mode can be activated, where all the valves are opened and pump and heater are deactivated to allow maintenance personnel to remove air from the system.

Climatisation comfort, enabling or disabling a room can be controlled by voice, or set via the google calendar or any one of the feeders app channels.

The climate system registers when doors or windows are open and stop normal climatisation to avoid energy loss.

Rooms can be prioritized when lots of rooms need energy (imagine coming home in the cold after a holiday, some rooms you need warming up faster than others).

Setting the house in Holiday mode disables all heating and maintains a safeguarding low (or high) temperature. Some rooms can deviate (a target setting exists for holiday mode) for example when you have plants that need upper or lower temperatures to survive.

The software takes care of the valves and pumps that they get active once a day to avoid malfunction due to not being used for a long period.

Reporting is very extensive, not only in real time on the displays of the climate controller(s), but also on a daily cumulative basis.

More information: **climate**.

Security App

Security is a backbone of any home control offering as it determines the house_occupancy mode (Away, Sleep, Day or Holiday). The home_occupancy status has major significance in the climate system, the lights and in notifications.

Much effort has been devoted to make the security management complete and the options and functionality can be a little overwhelming when presented the first time.

The reality is that sometimes a security system is already at the premises and through an interface Lucy has to be linked and integrated.

Therefore the setup of security systems within Lucy is possible in 3 scenario's:

- Lucy is a **slave** to another security system
- Lucy is **master** to another security system
- A **hybrid** situation is created whereby Lucy is master for certain alarms (say fire) and another security system for other alarms (burglar), they work intertwined.

If the name of an interface object starts with *_master __* as in 'master_r_fire', then it concerns a another security system as master, instructing Lucy to invoke a fire alarm.

If the name of an interface object starts with *_slave __* as in 'slave_s_fire', then it is Lucy that instructs a security slave system to report a fire situation.

As you notice in the table below, there are several master or slave prefixed interface objects, the former are obviously inputs, the latter are outputs

core name	what it does / how is it generated
fire	it concerns a fire alarm, generated by a Fire_detector, a master_r_fire input trigger or by triggering the do_fire virtual. When an fire alarm is reset, then it is ignored for the blackout_mode duration to allow the faulty detector or room ventilation to restore a non faulty situation
panic	a silent alarm is triggered, no sirens, this is generated by the triggering the do_panic virtual or by the master_r_panic input trigger
burglar	an alarm situation, generated by triggering the do_alarm virtual, an Alarm_detector or by the master_r_burglar input trigger
tamper	is the situation whereby someone tries to open a security box, cut a wire, etc.. It is generated by the do_tamper flag, the master_r_tamper input field

As you could read from the above table, there are do_xxxx virtuals, they trigger an alarm situation when they get active, but there are a few other virtuals with the same base name but with another prefix:

prefix	does what	examples

prefix	does what	examples
*do_*	'do' virtuals, they set the start of a condition when they become active	do_arm, do_arm_full_req, do_arm_full, do_arm_at_close, do_arm_partial, do_arm_partial_req
*is_*	'is' virtuals, they are active when a situation is true	is_alarm, is_alarm_eminent, is_arm_eminent, is_armed, is_armed_full, is_armed_partial, is_burglar, is_fire, is_holiday, is_panic, is_reboot, is_tamper
*o_*	are digital output things, active when a situation is true	o_alarm, o_armed, o_armed_full, o_armed_partial, o_burglar, o_fire, o_tamper, o_panic

2 Special virtuals are available for leaving armed or entering armed premises :

- **is_alarm_eminent** : when a door or an alarm is triggered in the **entry_way**, then this is ignored for a time (duration of the flag, typically 60 to 90 seconds) to allow the person to reach the device to stop arm and this pending alarm.
- **is_arm_eminent** : when a person arms the security system (do_arm_full_req) sets the arm, then when an alarm or a door is triggered in the **exit_way**, then this alarm is ignored for a time (duration of the flag, typically 60 to 90 seconds), until the system is then armed

entry_way and **exit_way** are members of safe_ways and are lists of doors and alarm_detectors that not immediately trigger an alarm (when leaving, exit_way, when entering entry_way) to allow to reach the exit or an alarm reset console.

As every room can be associated with a zone, the zones are then having security relevance:

- **ignore_zones**: rooms or spaces in these zones are ignored
- **always_zones**: zones that always have to be armed and secured when the system is armed
- **partial_zone**: zone to be armed only when do_arm_full

If you have more than one set of partial zones, f.i. in 2 cases whereby the residents are sleeping in the house or everyone is away but not the dog, then you can set the parameter partial_zone_index and use an index suffix for selecting the right partial zone.

In order for the system to be tested during setup, there is a maintenance_mode flag which allows to set the desired conditions without effectively making active the master outputs and sirens.

For details on **access_scenes**, see Access.

There are several lights that can be triggered when the system is armed, armed pending, fire, or alarm. Also a few notifications exist, where one is particular; When the system is in alarm, then an alarm_loop notification is triggered that can email a snapshot of every camera to have forensic evidence of what happened at the site.

There is a special app virtual **do_arm_at_close**. When this flag is high, and one of the doors in the exit_way closes, then the system is automatically immediately armed, and any arm_eminent delay is cut short. This is very handy to have a car leaving the premises and locking down the house when the garage door closes.

Finally there is extensive logging and reporting of security events, see an example below.

There has been an attempt to integrate the netatmo welcome system into Lucy, but for quality reasons of the netatmo door movement tags it has stayed separate.

Irrigation App

Irrigation for a vegetable garden or for other leisure plants is for some people an important aspect of their well being at home.

Many automatic irrigation systems exists, but they lack the comfort or unified usability such as here on offer. In my experience, these irrigation controllers are cumbersome to use, and typically fail to water just enough what is needed, either not enough or too much. Also in more complete setups you want to protect against hardware issues, such as pumps not stopping or broken pipes.

How does this app differs from other irrigation systems?

Firstly you can use your google calendar to set when you want that the irrigation process is triggered, alternatively you can determine to run at a fixed time.

Secondly, the time of watering at an irrigation point is determined with lots of brains and considerations, rather than a time that you need to change manually yourself.

Further there is frost protection built in, not only for your valves but also for your pump (if you use one).

Then of course you can trigger the irrigation with buttons, virtuals, using a feed through one of the feeder channels. Or skip the next irrigation point or just cancel all together a started irrigation.

You can define when are the winter months where you don't want irrigation.

Because irrigation can be supported easily by Artificial Intelligence, you could potentially add plant information, rain captured volume (in a silo), time of the year, and then determine when to water, how much.

So this is just the first version, but we are convinced that we will outsmart any competitors version easily.

More information: **irrigation**.

Weather Station App

Having a weather system that is linked with our Notifier App allows for nice scenario's. Imagine it rains for the first time in the day, you get a nice warm voice telling you that it is raining outside, making you feeling happy nonetheless.

Many sensors are supported: temperature sensors, rain sensors, rain gauge (measuring rain quantity), soil humidity, sun strength, wind speed, air quality, etc..

Many notifications exist linked to the sensors installed, for when it is freezing, thawing, or reaching very cold or very warm temperatures or warming up again after being very cold.

Weather forecast is retrieved from both darksky or wunderground or accu weather, and is available to the irrigation app.

It will scan the forecast on extreme wind, temperature or rain and issue a warning in a daily weather email briefing.

If actual sensors are used then actual extreme conditions are just notifications that you can prepare.

The weather system of netatmo is totally integrated into Lucy's weather system.

The netatmo_ws_driver integrates the netatmo weather station products. It reports not only on outside weather conditions, but also on indoor air quality (CO2) in the rooms where these sensors are placed.

Reporting is extensive and combining local sensors and netatmo statistics in one overview.

Utilities App

Consumption and production of utilities such as gas, water and electricity become every more important personal aspects in the dealing with climate change. Therefore, Lucy contains a simple, yet powerful app to manage and report on utilities with real time decision capabilities.

The minimum is to measure and report on utility production and consumption and for this a Meter thing and an Utility app are the basis. Obviously the capability of storage of the utility must exist, such as rain water in a tank, or electricity in a battery.

Utilities carry a rate that can be dependent on various factors such as the time of day, weekdays or even if the effect is coming in (f.i. buying electricity from the grid) or out (a different rate for electricity going in the grid).

Utilities can be defined with the usage and the intensity (kWh / kW for electricity or m3 / °C for hot water), and both factors must be comprehended.

Sometimes complex real time decisions have to be taken, imagine the case of heating an outdoor swimming pool either with a gas_boiler (requiring gas) or alternatively with a heat_pump (requiring electricity) and where the electricity can come from solar panels or from the grid. Obviously if the solar panels produce plenty, your batteries are full and the rate for producing for the grid is low, using the heat pump makes sense. Alternatively when it is ice-cold outside during the night (no solar and low heat-pump efficiency), then the gas_boiler is the best option. Anything between these extreme cases require careful consideration and automatic switching from one heating mode to another.

Another complex case exists where you need irrigation for your garden, and you need to select one of 3 potential sources: collected rain water (cheap but limited to you tank), ground water (pump and water quality) or purchased city water (expensive and not always possible because of restrictions during drought).

So an important factor is when to load or discharge your utility storage (battery)? When your car has a vehicle to grid capability, you don't want your car battery to be empty when you have to leave for a long trip.

How can this all be managed automatically without you having to resort to several phone apps, and your precious time being wasted on control? You want to live a life where the right things happen with your utilities and infrastructure while you can focus on living your life with the comfort of them instead?

See and be surprised what Lucy can do with utility management!

See Utilities for more information and examples.

Light Manager App

The light manager is responsible for driving the room or other lights in accordance with the value_logic that has been designed for each light. If a room is protected (secured by the security things_controller), then the room lights are automatically de-activated, unless a value_logic definition would state otherwise.

For that to work, the light_manager things_controller will use the drivers to Vera, Hue and IKEA bridges and gateways to set the lights to the values that are defined in value_logic. The security things_controller also has light control and drives the room lights off when the room is secured.

Lights can be of 3 types: Light() for on-off lights, Dim_light() and Color_light().

The light_manager emulates a Hue Bridge and therefore all the lights can be controlled from the Hue app that is widely available on mobile phones. Also all lights are voice controllable.

my_assistant App

Voice assistants such as Amazon Alexa, ok Google or Siri are wildly popular in the english spoken world, although german, french and japanese versions of amazon echo are being released. Nobody doubts that voice assistants have a bright future ahead of them, certainly when equipped with smart AI, resulting in a near human experience. And the future will certainly bring new ways of interacting with virtual assistants when hologram processing is perfected.

It is our believe that Lucy can become the smart home assistant earning a place in that world, integrating but not competing with the broader assistants that exist already.

my_assistant is a first version of such an integration and to achieve this, we have borrowed from the first philips hue bridge that was compatible with amazon echo.

The way this works is that Lucy becomes a multiple hue emulated bridge, one for climate commands, one for lights, and one for general commands.

The way it works is that you say to Amazon Alexa : please discover devices, and then suddenly all your rooms, lights and climate wishes becomes voice enable commands.

Then you can say, Alexa : please turn on the bedlight, please turn on the guest room,...

For ok_Google, it works a little different, but with the same result. ok_google commands are intercepted by IFTTT, and then brought to the security role things_controller who checks and then "calls" the right emulator and issues the command, all in the blink of an eye. If you say, ok google, turn on the bedlight, then she'll answer : ok honey, i ll pass the command.

For voice control with Apple Siri, a trick is used where by the user asks Siri to make a note which is then processed as a voice command Then you can say: "siri make note: turn on bedlight". This new note is fetched within a second and then played out.

Ensure you have yr gmail account with IMAP enabled and notes active, and link this account with your Apple Siri notes. The new notes will be fetched by the Google_driver and offered to My_assistant for processing.

There is however one catch. For the Hue emulation to work flawlessly, one need to install a real Hue Bridge. The cost cannot be a burden (i hope). All the hue emulator things controller have to get a link with this bridge and this is done by pressing the link button on the real bridge ones that emulator things controller is started up. Fortunately, this need to be done only once, and if you upgrade the software version of the real bridge, then the emulated bridges will automatically follow.

The upside is that you can use the very nice designed Hue App on your apple or android phone to link to these emulated bridges (due the linking within 5 minutes from booting the things_controller) and activate commands as you wish.

As a result, there is NO need for a Lucy mobile app, just use the Philips hue one.

Surely Philips do not object as you use one real bridge for all the emulated ones, and your hue lamps are obviously driven by the real one.

A more solid integration with the voice assistants of these suppliers can be arranged, especially if this is what many users want. They cannot block the voice thing to just what they support.

Things_Apps

Imagine you can specify somewhere what data is to be collected about every thing you have in the home.

Such data could be:

- ownership : maybe some of the things are not owned by yourself, but by the owner of the building
- utility_usage : some things use electricity, then you specify Watts, other things use water for irrigation, and then you have liters/minute
- an alternative label : one wire temperature sensors all look the same
- any other information need or app that will be released

With the above data you can estimate power and water consumption for all the things in scope because your home controller knows when they are active. Both apps Forensics and Utility_usage work together to get this overview accomplished.

Other properties could be more than data, say a sirens test associated with a siren thing.

This is what things_apps stands for, to allow you to define the data or app for your things or just one Thing.

The feeder

You want to have the possibility to trigger and switch things when you want and how you want.

Obviously for the geeks there is IFTTT (if this then that), but having a calendar where you can set the time for apps to start such as the irrigation for your plants is nice. But there could be other ways of starting something, f.i. when you send an SMS to lock down the house because you forgot to register your outgoing.

Enter the feeder, this is the "manager" and protector of all the channels who want to feed your home control with instructions to start an app or deliver a payload to a thing.

Several channels of input exist : via calendar, email, sms, tcp, udp but this will be expandable in the future to allow cloud apps (such as energy) to play a role and payload your things.

network_controller App

This is the first version of a full local network control concept.

Currently the network controller will read the ip address list from site.conf and ping every device to check it's presence on the network. Also the absence of internet access is periodically verified.

As a final 'network' related check, the mains power is checked and loss is reported.

Notifications can be issued when a device is lost or returns or when the internet is lost or returns.

In the current setup, every device needs to be clicked firm against the assigned ip address and ensured that this address is registered in the site.conf file, a very manual process not for everyone.

In future versions this should be automatic whereby these ip addresses are generated (dhcp service) and whereby the network is protected against attacks and performance issues.

Then the issue dissapears to manually make fix ip addresses linked into site.conf, so that name resolution to ip address happens without any burden to the user.

Plug in the new device and configure it online!

Vera Driver App

Vera Controls makes one of the most versatile gateways which are compatible with an extraordinary range of zwave, zigbee and bluetooth devices that this gateway can cover.

Instead of creating a big list of possible devices compatible with Lucy, better and smarter is to leverage on what exists within vera plus as it is already the broadest in the market.

When you purchase the veraplus gateway, make sure you obtain the zwave plus version for your region (US, Europe,..).

Create a free login at veracontrol http://getvera.com/controllers/veraplus and add all your devices, screens, curtains, etc.

Lucy contains some example scripts to trigger the feeder from scenes or to synchronize settings.

Define your veraplus as a Zwave() device in the raspi option in your Home Configuration File.

Any mobile app that works with vera can be used to make the right setup.
A good one to mention is VeraMate, available in the apple store.

Closing Comments

Is Universal Home Control for everyone or just for technology savvy people?

It is our believe and conviction that our approach can lead to a situation where everyone will use and enjoy it.

Ofcourse you need a super cool app to create and maintain the Portable Home Configuration File.

Ofcourse you need capable installers (just like for installing internet modems) who can help to put the solution in place.

Almost everyone was able to use the TV device from the operator to select channels and prepare recordings.

And most people will just change notifications and preferences, buy the things what deliver what they want and enjoy the outcomes.

Of critical importance is reliability, that has to be on par with the MBF of electricity provision or the home heating system. By being not dependent on the internet, this kind of reliability is achievable.

Most importantly, customers will see us as the providers of **ADDED_VALUE** on Things they have and will buy, not just a things provider with a gadget app to switch it on or off.

Our apps on the controller and in the cloud will unlock this added value big time.

No one else can offer this today, we can!

And home control is a must have in modern homes and command premiums in home valuation. Soon it is normal and expected that homes have app based controls for comfort and infrastructure.

Who has a solution that brings it all together will secure this opportunity.