PYTHA Kitchen

© Juni 2020 PYTHA Lab GmbH

Dear PYTHA customer,

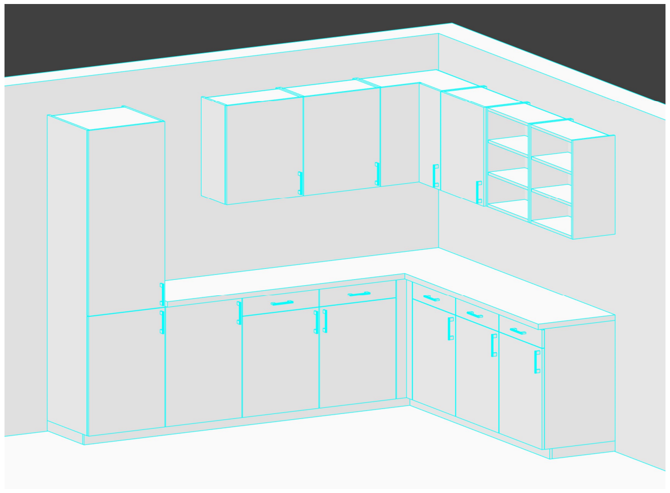
Welcome to the community of PYTHA users!

Your 3D CAD software PYTHA is one of the most powerful drawing and design systems in the market.

- Planning
- Presentation
- Production

PYTHA is the tool of your choice, easy to learn and highly efficient.

This manual is giving an introduction in the PYTHA Plug In 'Kitchen Wizard'.


Sample kitchen layout

Content

Introduction

This presentation will give you a guide to understanding and developing PYTHA plugins .

You will learn about the scope of the plugin architecture.
You will have the possibility to study several samples in detail.
And you will receive some relevant information around PYTHA plugins and version 25.

To use and develop plugins yourself, you need an installation and a license of PYTHA V25.

There is ample documentation available (see later).

If you have further questions, don’t hesitate to contact us.

All pictures and texts © PYTHA Lab GmbH

The Programming Language Lua

Why is PYTHA using Lua for its Plugins?

- Easy-to-learn programming language
- Friendly / familiar syntax (if, for, while, etc.)
- Documentation / tutorials / books readily available
- Powerful language and quite performant

To get started, see, e.g.

- https://www.lua.org/pil/contents.html
- https://www.amazon.com/exec/obidos/ASIN/8590379817/lua-pilcontents-20

PYTHA Lua Api Documentation

Where can I find a manual?

There is a github repository for the pytha lua api:
https://github.com/daniel-flassig/pytha-lua-api

It contains
- Sample Plugins
- A full reference documentation in the wiki
  https://github.com/daniel-flassig/pytha-lua-api/wiki

The documentation is constantly being updated for new functionality and improved with tutorials / samples, etc.

PYTHA Plugin Technology

What can the plugins do (already)?

- Custom buttons / functions for
  - Custom part types
  - Custom generators
  - Custom tools functions
  - Custom dialogs for existing functionality

- Custom edit functionality (e.g. for generator-created parts)

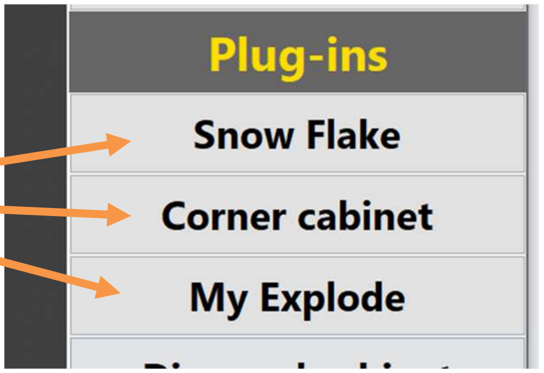
- Custom part filters in the part selection

- Custom attributes
  - Custom data field per part
  - Custom calculation for the parts list / reporting

Custom Buttons: Function Extensions

Build new generators / custom part types / custom tools

Represented by a button in the user interface



Highlights:

- May display a dialog (later)
- Interactive
- Create standard parts
- Create parts from facets (arbitrary geometry)
- Modify parts
- Change attributes
- Create groups / ngos
- Import library pyos (with parametrics)

Custom Buttons: Function Extensions

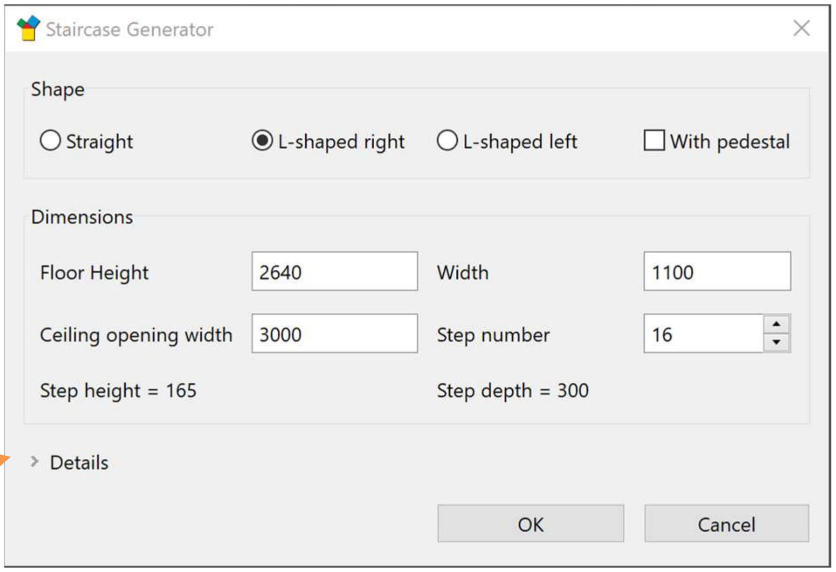
Design User Interface / Dialogs

Easy to create in Lua code

Highlights:

- Familiar user-interface elements
- Automatic layout(!)
- Interactive response
- Group boxes that can unfold

- Easy to translate / localize



Staircase Generator

Shape

( ) Straight (●) L-shaped right ( ) L-shaped left ☐ With pedestal

Dimensions

Floor Height	2640	Width	1100
Ceiling opening width	3000	Step number	16
Step height = 165		Step depth = 300	

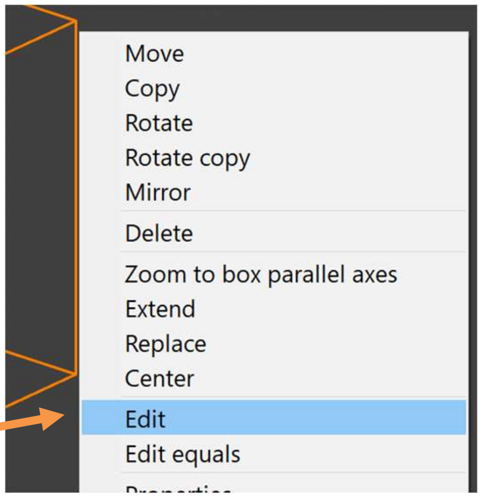
Right-Click Edit: Edit Extensions

Provide Edit capabilities with your plugin generators

Allow the user to right-click and press Edit for a plugin-created part

Highlights:

- Parts and Groups can remember the input that was used to create them (history)
- Arbitrary information can be stored
- The standard “Edit” function works
- The same plugin user interface can be invoked again



Part Filters: Part Selector Extension

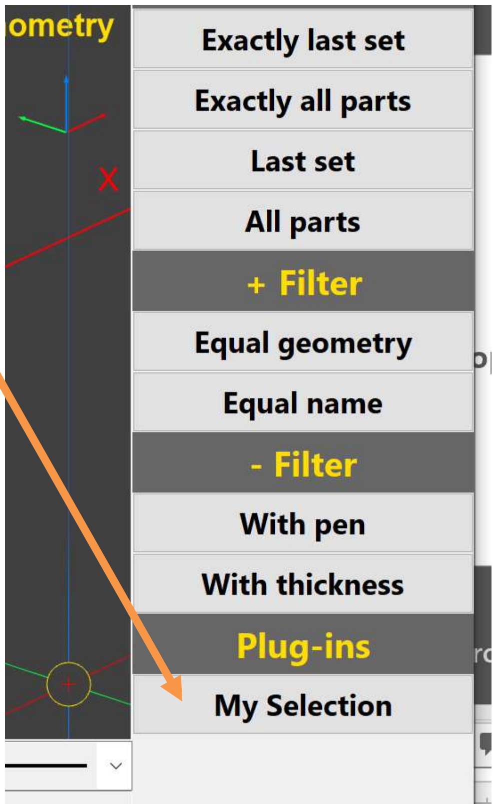
Implement custom part filters

The custom filters are displayed in the selection menu for all functions that use parts

- e.g. tools, parts list, cam output

Highlights:

- Use all part attributes
- Filter by multiple criteria
- Display dialogs to adjust the filters



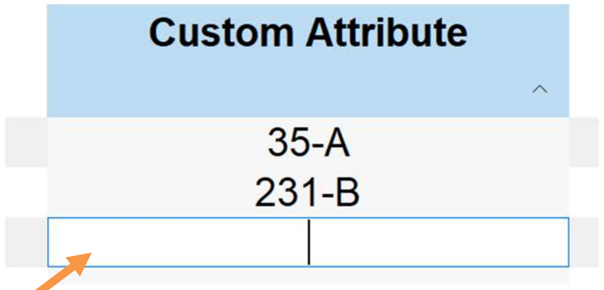
Additional Info: Attribute Extensions

Attach new data fields to elements

Previously only 6 user-attributes available.
Collisions possible with import!

Now: unlimited custom attributes for parts, groups and ngos.

Highlights:
- Displayed and editable in the parts list
- Will be displayed in the attributes dialog (not yet)
- Automatically saved and loaded with the pyo
- No problem during import of foreign pyo
- As many new attributes as you can think of!
- Long texts are not a problem

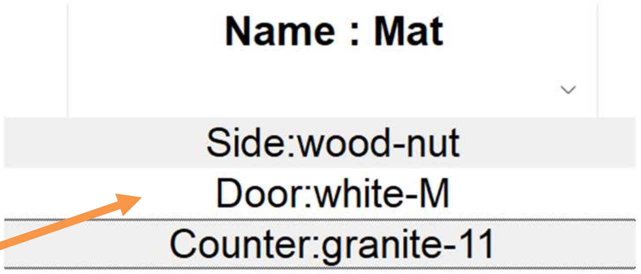


Custom Attribute
35-A
231-B

Additional Info: Attribute Extensions

Evaluate new quantities in the parts list

Highlights:
- Can use all built-in attributes
- Can use plugin attributes as well
- Concatenate text (e.g. Name : Material)
- Evaluate complex logics



Name : Mat
Side:wood-nut
Door:white-M
Counter:granite-11

The Plug In ‘Kitchen Wizard’

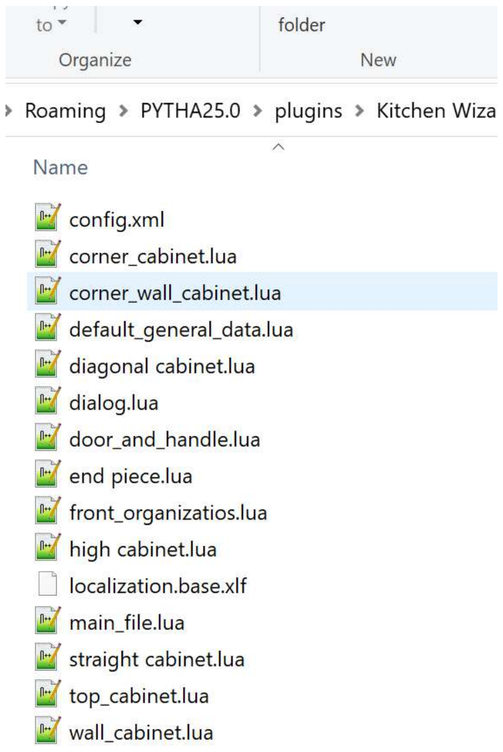
What is the PYTHA Kitchen Wizard?

- Intuitive and easy wizard/generator for configuring complete kitchens
- Extendable architecture to host wardrobe/workspace/bathroom solutions
- Cabinets are automatically arranged, different settings can either affect all cabinets (e.g. board thickness) or individual cabinets (e.g. width)
- Front styles can be exchanged
- Appliances will be added
- Library elements can be included
- Cooking islands?

The Plug In ‘Kitchen Wizard’

Central files

- *main_file.lua*: Main function, edit function, cabinet positioning logic
- *dialog.lua*: Dialog creation and navigation handling, neighborship bookkeeping
- *default_general_data.lua*: Default values and sorting of cabinets
- *straight_cabinet.lua*: Well commented, simple cabinet as basis for custom cabinets
- *config.xml*: Definition of entry point, buttons in menu and edit function. **Use own GUID!**
- *localization.base.xliff*: Be international – translate!
- *front_organizations.lua*: Examples for different fronts
- *door_and_handle.lua*: Basic handles and doors

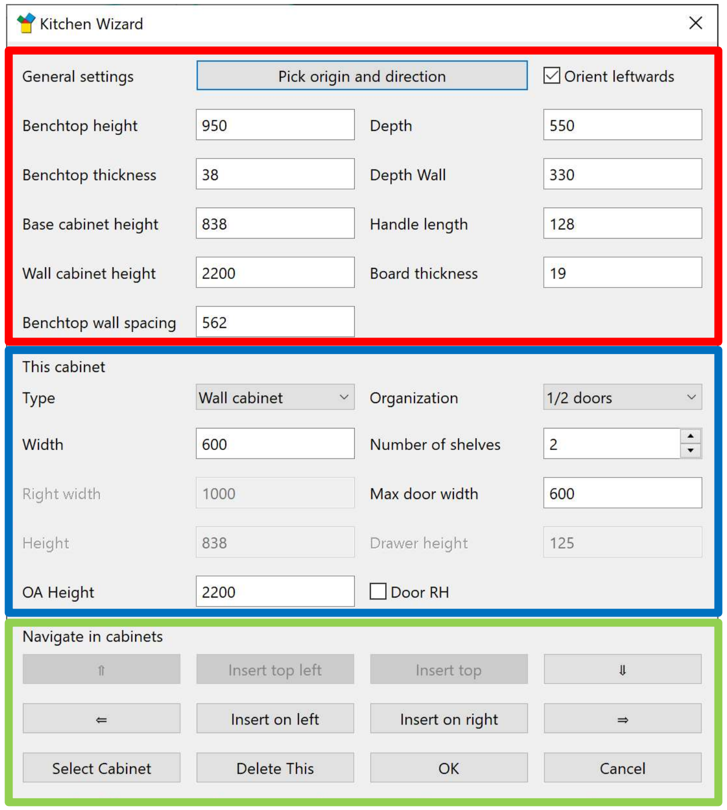


config.xml
corner_cabinet.lua
corner_wall_cabinet.lua
default_general_data.lua
diagonal cabinet.lua
dialog.lua
door_and_handle.lua
end piece.lua
front_organizatios.lua
high cabinet.lua
localization.base.xlf
main_file.lua
straight cabinet.lua
top_cabinet.lua
wall_cabinet.lua

The Plug In 'Kitchen Wizard'

The Wizard Dialog

- Consists of 3 sections
  - General kitchen settings
  - Cabinet specific settings
  - Navigation and Insertion
- Cabinet specific settings can be modified and disabled depending on cabinet and front requirements
- General settings can be customized for application requirements



Kitchen Wizard

General settings — Pick origin and direction — ☑ Orient leftwards

Benchtop height	950	Depth	550
Benchtop thickness	38	Depth Wall	330
Base cabinet height	838	Handle length	128
Wall cabinet height	2200	Board thickness	19
Benchtop wall spacing	562		

This cabinet

Type	Wall cabinet	Organization	1/2 doors
Width	600	Number of shelves	2
Right width	1000	Max door width	600
Height	838	Drawer height	125
OA Height	2200	☐ Door RH	

Navigate in cabinets

The Plug In ‘Kitchen Wizard’

dialog.lua

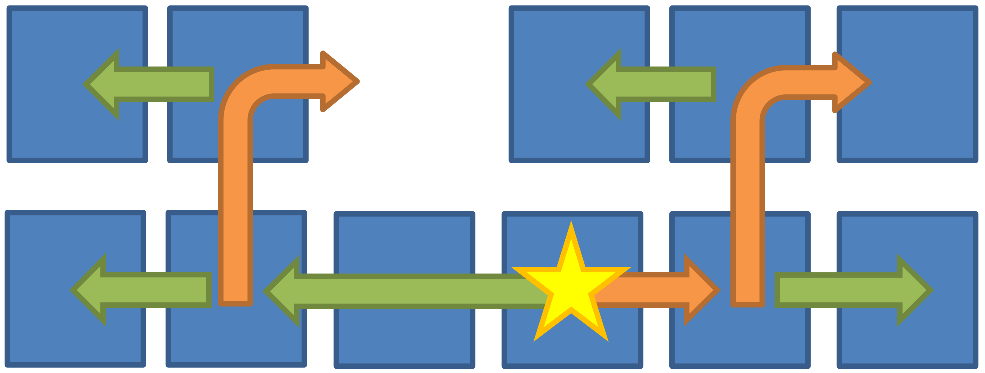
- Function wizard_dialog(dialog, data): initialize dialog
- Soft_update: mostly used to enable Door RH check
- Unicode characters ("\u{21D1}“) can be used as arrows
- *controls* contains all controls that need to be accessed during runtime
- Rows: *0x1* base (first bit), *0x2* top (second bit), *0x3* high (both bits)

Top row has right_top and left_top elements, base has right and left and high can have all of them

The Plug In ‘Kitchen Wizard’

main_file.lua

- function *main* and function *edit_wizard*
- *create_geometry_for_element* calls
  subgroup = *spec_type_info.geometry_function*
- *recreate_geometry* iterates over all cabinets
- Iteration order: orange first, green afterwards



The Plug In ‘Kitchen Wizard’

default_general_data.lua

- *general_default_data* and *initialize_cabinet_values*: modify to typical values!

- *cabinet_sorting_for_combobox*: defines the order in the type drop list

- *init_typecombolist*: sorts the *cabinet_typelist* into order and type
- *assign_cabinet_type*: puts values from *cabinet_typelist[type]* to *specific_data*

The Plug In ‘Kitchen Wizard’

straight_cabinet.lua

- Lines 1..115 create dialog for straight cabinet, **can be omitted**

- *recreate_straight*: Do the geometry, group all parts
    - Return the handle to the group!
    - *specific_data.elem_handle_for_top* used for benchtop
        - Cutouts possible!
    - Kickboard handles L/R will be booled, others put in *general_data.kickboards* (check diagonal_cabinet)
- *organization_style_list[specific_data.front_style]. geometry_function*
    - Creates fronts depending on selected type

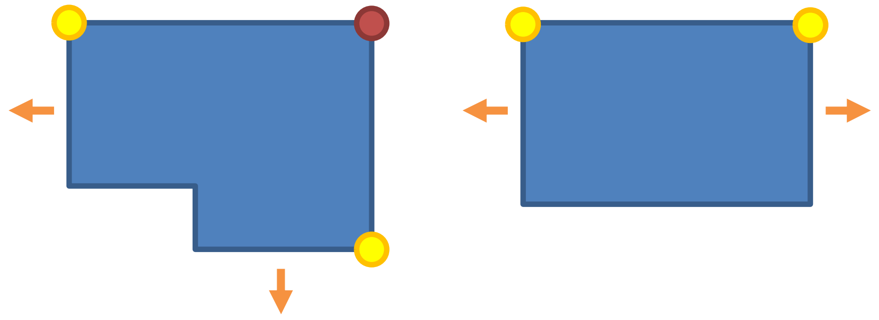
- Geometry only needs 100-200 lines of code

The Plug In ‘Kitchen Wizard’

straight_cabinet.lua

- *ui_update_straight*: Activate necessary *controls*, deactive by default
- Only activate upon soft_update==false to reduce flickering

- *placement_straight*: Define connection points (like reference points)
    - right_connection_point, left_connection_point
    - right_direction, left_direction (check corner_cabinet)
    - [origin_point] optional for corner cabinets
    - They define the relative positioning of connecting cabinets



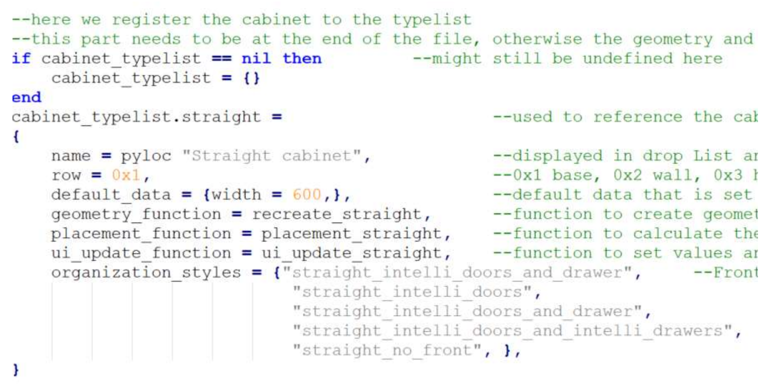
The Plug In ‘Kitchen Wizard’

straight_cabinet.lua

- Essential part: *cabinet_typelist.straight* = {…}
- Automatically inserts the cabinet into list

- For sorting: put the key (*“straight”*) into *cabinet_sorting_for_combobox* in position

- The front styles for this cabinet can be defined (in order) in *organization_styles*
    - Key for *organization_style_list*



```
--here we register the cabinet to the typelist
--this part needs to be at the end of the file, otherwise the geometry and
if cabinet_typelist == nil then         --might still be undefined here
    cabinet_typelist = {}
end
cabinet_typelist.straight =                          --used to reference the ca
{
    name = pyloc "Straight cabinet",                 --displayed in drop List ar
    row = 0x1,                                       --0x1 base, 0x2 wall, 0x3
    default_data = {width = 600,},                   --default data that is set
    geometry_function = recreate_straight,           --function to create geomet
    placement_function = placement_straight,         --function to calculate the
    ui_update_function = ui_update_straight,         --function to set values ar
    organization_styles = {"straight_intelli_doors_and_drawer",      --Front
                        "straight_intelli_doors",
                        "straight_intelli_doors_and_drawer",
                        "straight_intelli_doors_and_intelli_drawers",
                        "straight_no_front", },
}
```

The Plug In ‘Kitchen Wizard’

front_organizations.lua and door_and_handle.lua

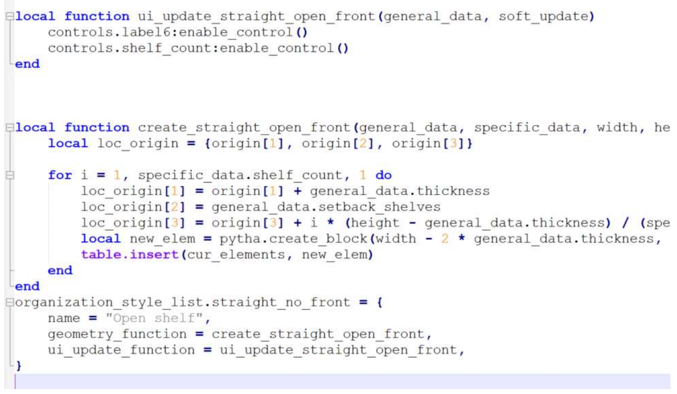
Different front and organization styles can be assigned to one cabinet.

- *organization_style_list*: Contains all styles
    - Each cabinet will filter its possible styles

- Each front needs a *ui_update_function*
    - Can enable drawer box height etc

- *geometry_function* with varying arguments
    - Width, height of door section
    - Space for shelves, pullouts, carousels
    - Created elements need to be grouped outside.
        - Either table as argument or return a table

- Logic for door handles

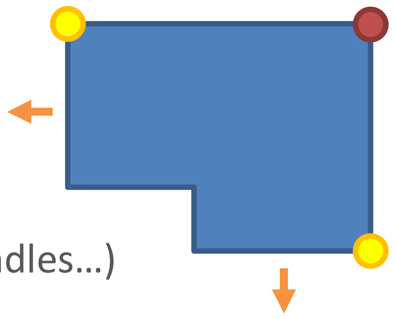


```lua
local function ui_update_straight_open_front(general_data, soft_update)
    controls.label6:enable_control()
    controls.shelf_count:enable_control()
end


local function create_straight_open_front(general_data, specific_data, width, he
    local loc_origin = {origin[1], origin[2], origin[3]}

    for i = 1, specific_data.shelf_count, 1 do
        loc_origin[1] = origin[1] + general_data.thickness
        loc_origin[2] = general_data.setback_shelves
        loc_origin[3] = origin[3] + i * (height - general_data.thickness) / (spe
        local new_elem = pytha.create_block(width - 2 * general_data.thickness,
        table.insert(cur_elements, new_elem)
    end
end
organization_style_list.straight_no_front = {
    name = "Open shelf",
    geometry_function = create_straight_open_front,
    ui_update_function = ui_update_straight_open_front,
}
```

The Plug In ‘Kitchen Wizard’



Adding new geometries

- **Library items**: 3 or 4 ref points for placement
- pyux.select_pyo and pytha.import_pyo
- Not yet all elements in folder as a table (for drop list of handles…)

- **New geometries**: adapt an example to your needs
  - Extractor hood

Appendix

Some useful Definitions

api	Application programming interface – here: functions that a plugin can invoke
config.xml	Configuration file that describes the identity and functionality of a plugin
Extension	A single capability inside PYTHA that is provided by a plugin
github	An online platform where you can find the documentation for the plugin api
Lua	Simple, yet powerful scripting language used to develop PYTHA plugins
Plugin	Package to customize PYTHA which may contain one or more extensions
xlif	A standard localization file format used to translate the plugins

Notes

Have fun and success
with the **PYTHA Kitchen Wizard!**

Daniel & Fabian Flassig,
September 2020