NKA Maker

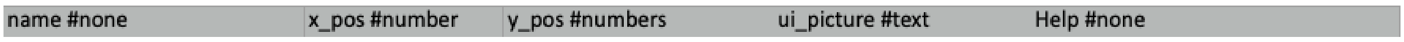
A python tool that serves for generating .nka files from .csv tables.
The column's number/size/content/type can be fully customized according to needs.

Creating the the .csv file(s):

If you're working on mac with "numbers", simply go to Files -> Export to -> CSV, to create a .csv file from the .numbers file

- You can create a table with any values/parameters that you'd like to turn into an .nka file.
- Multiple .csv files can be used in 1 project, to allow more modularity and clarity
- 1 .nka file will be generated for each column
- The generated .nka file will have the following name: csv_filename_column_name, e.g: fx_controls_x_pos.nka
- When creating your table in the .csv file, the first row should always be reserved for:
    - Column Name. e.g: x_pos , y_pos, ui_pictures …
    - Column Type: Type of the column content:
        - **#text:** will generate the .nka file as a text array
        - **#int:** will generate the .nka file as a int array
        - **#real:** will generate the .nka file as a real array
        - **#none**: will ignore the column, and won't generate any .nka files
    - Column Name and Type should follow the following naming convention:
      column_custom_name #type
      e.g: *x_pos #numbers*
      e.g: *y_pos #numbers*
      e.g: *ui_picture #text*
      e.g: *help_messages #none*

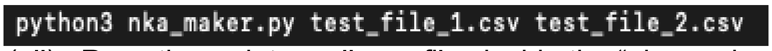
Example of first row:



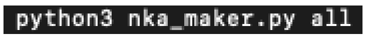
name #none	x_pos #number	y_pos #numbers	ui_picture #text	Help #none

Steps:

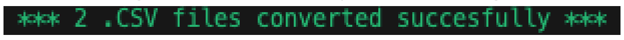
- In the instrument's root folder (where the .nki file is), create a folder named "nka_maker".
- inside of the "nka_maker" folder you should have:
    - the .csv files that needs to be processed into .nka files
    - the python script: "nka_maker.py"
- run the python script from the terminal. The script can be run in 2 modes:
    - Mode 1 (arguments) : Runs the script on the .csv files entered as arguments

      ```
      python3 nka_maker.py test_file_1.csv test_file_2.csv
      ```
    - Mode 2 (all) : Runs the script on all .csv files inside the "nka_maker" folder

      ```
      python3 nka_maker.py all
      ```
- If processing is successful, you should get the following message:

  ```
  *** 2 .CSV files converted succesfully ***
  ```
  And the .nka files will be generated in: *Resources/data/*
  If any issues happen while processing due to wrong type, wrong format, non-existing files…
  A self explanatory error message will be displayed in the terminal

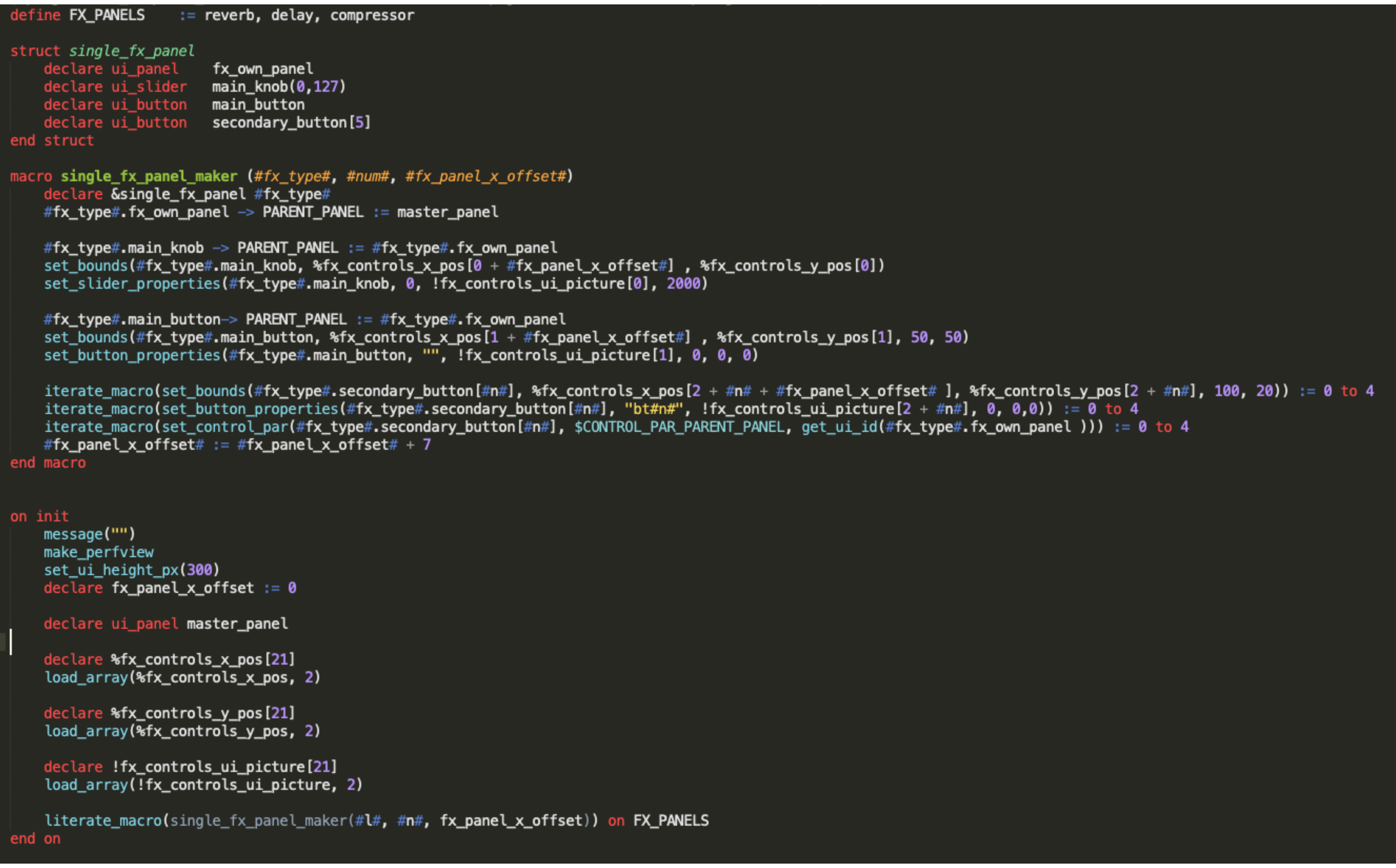
EXAMPLE: A Simple application for implementing 3 ui groups that share the same layout.

.CSV file:

name #none	x_pos #number	y_pos #numbers	ui_picture #text	Help #none
rev_drywet	30	15	knob_mini	Test text
rev_onoff	65	18	on_off_button	Test text
rev_settings_1	30	40	on_off_button	Test text
rev_settings_2	30	60	on_off_button	Test text
rev_settings_3	30	80	on_off_button	Test text
rev_settings_4	30	100	on_off_button	Test text
rev_settings_5	30	120	on_off_button	Test text
delay_drywet	150	15	knob_mini	Test text
delay_onoff	185	18	on_off_button	Test text
delay_settings_1	150	40	on_off_button	Test text
delay_settings_2	150	60	on_off_button	Test text
delay_settings_3	150	80	on_off_button	Test text
delay_settings_4	150	100	on_off_button	Test text
delay_settings_5	150	120	on_off_button	Test text
compressor_drywet	270	15	knob_mini	Test text
compressor_onoff	305	18	on_off_button	Test text
compressor_settings_1	270	40	on_off_button	Test text
compressor_settings_2	270	60	on_off_button	Test text
compressor_settings_3	270	80	on_off_button	Test text
compressor_settings_4	270	100	on_off_button	Test text
compressor_settings_5	270	120	on_off_button	Test text

This will generate 3 .nka files:
fx_controls_x_pos
fx_controls_y_pos
fx_controls_ui_picture

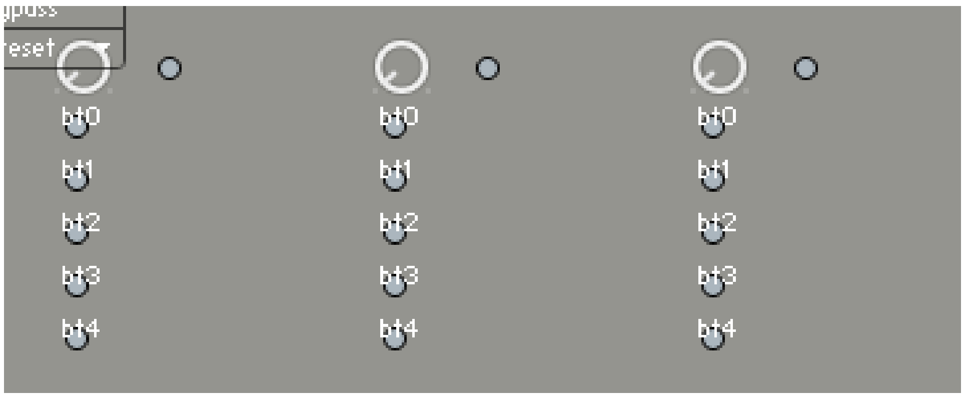
KSP Declaration/Implementation:



```
define FX_PANELS    := reverb, delay, compressor

struct single_fx_panel
    declare ui_panel    fx_own_panel
    declare ui_slider   main_knob(0,127)
    declare ui_button   main_button
    declare ui_button   secondary_button[5]
end struct

macro single_fx_panel_maker (#fx_type#, #num#, #fx_panel_x_offset#)
    declare &single_fx_panel #fx_type#
    #fx_type#.fx_own_panel -> PARENT_PANEL := master_panel

    #fx_type#.main_knob -> PARENT_PANEL := #fx_type#.fx_own_panel
    set_bounds(#fx_type#.main_knob, %fx_controls_x_pos[0 + #fx_panel_x_offset#] , %fx_controls_y_pos[0])
    set_slider_properties(#fx_type#.main_knob, 0, !fx_controls_ui_picture[0], 2000)

    #fx_type#.main_button-> PARENT_PANEL := #fx_type#.fx_own_panel
    set_bounds(#fx_type#.main_button, %fx_controls_x_pos[1 + #fx_panel_x_offset#] , %fx_controls_y_pos[1], 50, 50)
    set_button_properties(#fx_type#.main_button, "", !fx_controls_ui_picture[1], 0, 0, 0)

    iterate_macro(set_bounds(#fx_type#.secondary_button[#n#], %fx_controls_x_pos[2 + #n# + #fx_panel_x_offset# ], %fx_controls_y_pos[2 + #n#], 100, 20)) := 0 to 4
    iterate_macro(set_button_properties(#fx_type#.secondary_button[#n#], "bt#n#", !fx_controls_ui_picture[2 + #n#], 0, 0,0)) := 0 to 4
    iterate_macro(set_control_par(#fx_type#.secondary_button[#n#], $CONTROL_PAR_PARENT_PANEL, get_ui_id(#fx_type#.fx_own_panel ))) := 0 to 4
    #fx_panel_x_offset# := #fx_panel_x_offset# + 7
end macro


on init
    message("")
    make_perfview
    set_ui_height_px(300)
    declare fx_panel_x_offset := 0

    declare ui_panel master_panel

    declare %fx_controls_x_pos[21]
    load_array(%fx_controls_x_pos, 2)

    declare %fx_controls_y_pos[21]
    load_array(%fx_controls_y_pos, 2)

    declare !fx_controls_ui_picture[21]
    load_array(!fx_controls_ui_picture, 2)

    literate_macro(single_fx_panel_maker(#l#, #n#, fx_panel_x_offset)) on FX_PANELS
end on
```

- fx_panel_x_offset is used to jump to the next panel's array index after implementing the first panel.
- used: structs, macros, literate/iterate macros and ui arrays

UI Result:



ypass
reset
bt0
bt1
bt2
bt3
bt4
bt0
bt1
bt2
bt3
bt4
bt0
bt1
bt2
bt3
bt4