Version: ver. 1.5.4

Managing Assets on External Drive

Assets from drives/partitions

Until now the images were converted to `C` arrays that were compiled/hardwired into the executable code. Sometimes it's easier on ROM resources and more flexible to load images from files at runtime, making the smaller programcode uploads faster. This is possible now because LVGL function `lv_img_set_src` can set an array in RAM too that was loaded from a file, or it can even receive a filename of an image/picture-file as a parameter.

Exportable formats

Now these image-formats can be exported from SquareLine Studio (selectable in Project Settings):

- `Source-code`: exporting the images in C array and compiling them directly into the executable file (the way it was done until now)
- `Binary raw`: exporting the images in the binary pre-compiled version, that would otherwise be compiled from the source-code version.
- `Binary compressed`: Using a fast LZ compression algorithm it reduces the image sizes on the disk/partition significantly.
- `Binary LVGL`: exporting the pictures in PNG picture format that LVGL can handle (and manage/cache in the memory automatically).

Usage of exported assets

The image/picture-file to load can reside at different places depending on the selected LVGL `fsdrv` drive-type:

- For `STDIO`, `POSIX`, `WIN32` simulated drives/partitions a `drive` folder is created in the exported code, its subfolder called `assets` contains the images
- For `FATFS` (and `LittleFS` supported by LVGL-8.3.11) real drives the dedicated flash-partition or an external SD-card/etc can hold the images. You need to copy `assets` folder from `drive` folder into the real partition.

(You can give a different root folder in Project Settings to register a folder other than `drive` for your STDIO/POSIX/WIN32 filesystem-driver, and that is set in `lv_conf.h`.)

Loading the images as `Binary LVGL` is simply done by the above mentioned `lv_img_set_src`, but not adding a pointer of an `lv_img_dsc_t` structure as an argument, but a path+filename. (LVGL knows that it's a path because its first byte is an ASCII character in the range of 0x20..0x7F)

Loading the image data into RAM for `Binary raw` and `Binary compressed` formats are done by `UI_LOAD_IMAGE` function of `ui_img_manager.h` (only created for binary image formats). (The raw format uses `_ui_load_binary` function that needs 2 arguments: filename, size. The compressed format is extracted to RAM and occupies the same amount of memory (after being uncompressed) as the raw format. It uses `_ui_load_compressed_binary` function in the background that takes 3 arguments: filename, compressed size and normal size.) Loading the image data is only the first part of the operation, you need to create an `lv_img_dsc_t` structure and set its `data` member to tho pointer to the loaded data, and fill the other parts of the struct with sensible values. When the `lv_img_dsc_t` struct is complete, only then you can register it to the image widget by `lv_img_set_src` function. If all went well, the image will be displayed.

(Note: The filenames should contain the `assets` folder in the path, the root path in `lv_conf.h` should contain the `drive/` folder in a default setup to work.)

Example

If you imported a 128x128-pixel 65536byte RGB image called `picture.png` (format of SquareLine Studio icon) into SquareLine Studio assets, added it into the screen `Screen1`, and set the image-export format as `Binary raw`, selected `C:` letter for `STDIO` drive, your export will create:

- a `drive` folder beside `ui` folder with the `assets` subfolder and in it the exported image-pixeldata `ui_img_picture_png.bin`
- `lv_conf.h` has `LV_USE_FS_STDIO` set to 1, `LV_FS_STDIO_LETTER` to `'C'`, `LV_FS_STDIO_PATH` to `"drive/"`
- in `ui` folder the `images` subfolder has `ui_img_picture_png.c` with the content:
  - the struct for the image: `lv_img_dsc_t ui_img_picture_png = { .header.always_zero = 0, .header.w = 128, .header.h = 128, .header.cf = LV_IMG_CF_TRUE_COLOR };`
  - loading the data to struct with: `ui_img_picture_png.data = UI_LOAD_IMAGE( "C:assets/ui_img_picture_png.bin", 65536 );` and `ui_img_picture_png.data_size = 65536;`
- in `ui` folder the `screens` subfolder has `ui_Screen1` with the init function `void ui_Screen1_screen_init ()` (called by `ui_init` function in `ui.c`):
  - creating the screen: `ui_Screen1 = lv_obj_create(NULL);`
  - creating the image on it: `ui_Image1 = lv_img_create(ui_Screen1);`
  - selecting the source of the image: `lv_img_set_src(ui_Image1, &ui_img_picture_png);`

These were the steps to load and display an image from a file asset.

There are three main parts of the program: Header Menu, Panel windows and Editable View. In editor mode, panels can be positioned and measured freely to create a work space of your choice.