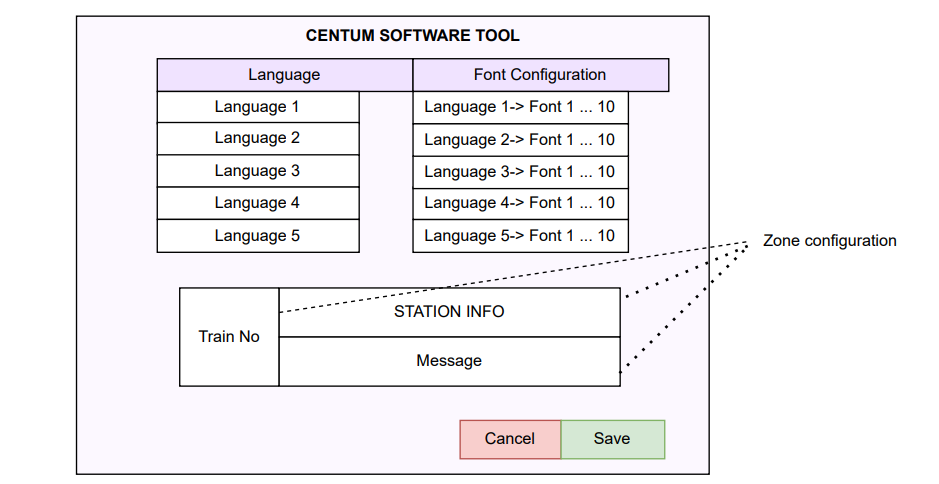
## Software Tool

The tool runs on PC. It is used to configure the LED font including bitmaps, icons and zones. It utilizes the icons, fonts that are stored in the PC. The generated file will be a C-Array file(binary).



The C array file will be an input for the font handling in the rendering engine.

It will support up to 10 configurations of each of the mentioned languages.

* English, Hindi, Marathi, Kannada, Malayalam, Tamil, Telugu, Gujarati, Punjabi, Bengali, Oriya and Urdu.
* LED display can be configured to support up to 5 languages, it’s based on the memory constraint of the NXP controller.

Step-by-step prerequisites to run the provided Python GUI code for TTF to C array conversion:

1. **Install Python 3.x**
   * Download and install Python 3.x from the official website.
   * Verify installation by running python --version in a terminal or command prompt.
2. **Set up Python Project (e.g., PyCharm)**
   * Create a new Python project.
   * Configure project interpreter to Python 3.x.
3. **Install Required Libraries**
   * Open a terminal or PyCharm terminal.
   * Install Pillow:

text

pip install pillow

* + Ensure Tkinter is available:
    - Tkinter is included by default in most Python installations.

python -m tkinter – terminal\

* + - On some Linux systems, you may need to install it separately, e.g.:

text

sudo apt-get install python3-tk

1. **Prepare Font File**
   * Have a TrueType font file (.ttf) accessible on your system.
   * This file will be selected later via the GUI.
2. **Run the Python Script**
   * Run the script containing the provided code.
   * A GUI window opens for selecting the TTF file.
   * After selection, click the button to generate the C array file.
3. **Check Output**
   * The generated C file named font\_data.c will be saved in the same directory.
   * Verify that the file contains the expected C array data.

This ensures the environment and necessities for executing the font converter GUI application are met.  
  
python code  
complete Python GUI code for converting a TrueType Font (.ttf) file to a C array file, named font\_data.c:  
  
import tkinter as tk

from tkinter import filedialog, messagebox

from PIL import Image, ImageDraw, ImageFont

import os

# Constants

CHAR\_START = 32 # Space

CHAR\_END = 127 # ASCII printable characters

IMAGE\_SIZE = (16, 16) # Width x Height for each character

# GUI Application

class FontConverterApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("TTF to C Array Font Converter")

self.root.geometry("500x250")

# UI Elements

self.label = tk.Label(root, text="Select a TrueType Font (.ttf) file:")

self.label.pack(pady=10)

self.font\_path\_entry = tk.Entry(root, width=50)

self.font\_path\_entry.pack(pady=5)

self.browse\_button = tk.Button(root, text="Browse", command=self.browse\_ttf)

self.browse\_button.pack(pady=5)

self.generate\_button = tk.Button(root, text="Generate C Array", command=self.generate\_c\_array)

self.generate\_button.pack(pady=20)

def browse\_ttf(self):

file\_path = filedialog.askopenfilename(filetypes=[("TTF Files", "\*.ttf")])

if file\_path:

self.font\_path\_entry.delete(0, tk.END)

self.font\_path\_entry.insert(0, file\_path)

def generate\_c\_array(self):

font\_path = self.font\_path\_entry.get()

if not font\_path or not os.path.isfile(font\_path):

messagebox.showerror("Error", "Please select a valid .ttf file.")

return

try:

font\_size = 16

font = ImageFont.truetype(font\_path, font\_size)

output\_lines = []

output\_lines.append("unsigned char font\_data[] = {\n")

for char\_code in range(CHAR\_START, CHAR\_END):

char = chr(char\_code)

img = Image.new("1", IMAGE\_SIZE, 0) # mode '1' = 1-bit pixels

draw = ImageDraw.Draw(img)

draw.text((0, 0), char, font=font, fill=1)

output\_lines.append(f" /\* '{char}' ({char\_code}) \*/ ")

for y in range(img.height):

row\_byte = 0

for x in range(img.width):

pixel = img.getpixel((x, y))

row\_byte = (row\_byte << 1) | (1 if pixel else 0)

output\_lines.append(f"0x{row\_byte:02X}, ")

output\_lines.append("\n")

output\_lines.append("};\n")

# Save to file

output\_filename = "font\_data.c"

with open(output\_filename, "w") as f:

f.writelines(output\_lines)

messagebox.showinfo("Success", f"C array generated as '{output\_filename}'.")

except Exception as e:

messagebox.showerror("Error", f"An error occurred: {str(e)}")

# Run the GUI application

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = FontConverterApp(root)

root.mainloop()

**Steps to Generate .c file in LVGL Format :**

To generate an LVGL-style C font file like "Mangal-Regular.c" from a TTF font in Python, the code must:

* Render each glyph bitmap and store as hex bytes in one large bitmap array
* Extract glyph metrics like advance width, bounding box size & offsets
* Calculate bitmap indices for each glyph
* Format the output with header, glyph bitmap array, glyph description array, charmap, and font descriptor structs

1. I'll generate a Python script sample that uses Pillow and FreeType to achieve this:  
     
   **Python 3.x**  
   Ensure Python 3.x is installed. Verify with:

text

python --version

1. **Required Python Packages**  
   Install necessary packages via pip:

text

pip install freetype-py

Note: tkinter is typically included with Python, but on some Linux distributions, you may need to install it separately (e.g., sudo apt-get install python3-tk).

1. **TrueType Font File (.ttf)**  
   Have your .ttf font file ready and accessible for selection in the GUI.
2. **Running Environment**  
   You can run this script in any Python environment: terminal, command prompt, or an IDE like PyCharm.

**Step-by-step Guide**

1. **Save the Provided Python Script**  
   Copy and paste the entire Python code into a file named, for example, font2c\_lvgl.py.
2. **Run the Script**  
   Open terminal or your IDE's run option and execute:

text

python font2c\_lvgl.py

This opens a GUI window for font selection.

1. **Select the TTF File**  
   Click the "Browse Font File" button, select your .ttf file, and it will show the file path in the input box.
2. **Generate LVGL C Font File**  
   Click "Generate LVGL .c Font File" button to start the conversion.
   * The .c font file is generated in the same directory as the chosen .ttf file.
   * If a file with the same name already exists, a unique filename with appended (1), (2), etc., is created automatically.
3. **Check the Output**  
   The script shows a success dialog and outputs the path of the generated .c file.  
   You can directly include this C source file in your LVGL project.
4. **Troubleshooting**
   * Make sure the .ttf file path is valid and accessible.
   * Ensure you have proper permissions to write into the directory containing the .ttf file.
   * Install freetype-py properly as it is necessary for font processing.
5. **Optional: Convert to .exe for Distribution**  
   To create a standalone Windows executable for this, install PyInstaller:

text

pip install pyinstaller

Then create exe:

text

pyinstaller --onefile --windowed font2c\_lvgl.py

Executable will be in the dist folder.

**Python code :**

This Python GUI allows users to select a TrueType font and generates an LVGL-compatible C font source file saved beside the input file.  
It extracts glyph bitmaps and metrics to produce a properly formatted LVGL C font file with comments and structure.  
If a file with the same name exists, it creates a uniquely named file to avoid overwriting.  
  
import tkinter as tk  
from tkinter import filedialog, messagebox  
import os  
import freetype  
  
class LVGLFontConverterApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("TTF to LVGL C Font Converter")  
 self.root.geometry("700x250")  
  
 self.label = tk.Label(root, text="Select a TrueType Font (.ttf) file:")  
 self.label.pack(pady=10)  
  
 self.entry = tk.Entry(root, width=80)  
 self.entry.pack(pady=5)  
  
 self.browse\_btn = tk.Button(root, text="Browse Font File", command=self.browse\_file)  
 self.browse\_btn.pack(pady=5)  
  
 self.convert\_btn = tk.Button(root, text="Generate LVGL .c Font File", command=self.convert\_font)  
 self.convert\_btn.pack(pady=20)  
  
 self.status = tk.Label(root, text="", fg="green")  
 self.status.pack()  
  
 def browse\_file(self):  
 filename = filedialog.askopenfilename(filetypes=[("TTF Font Files", "\*.ttf")])  
 if filename:  
 self.entry.delete(0, tk.END)  
 self.entry.insert(0, filename)  
  
 def get\_unique\_filename(self, path):  
 base, ext = os.path.splitext(path)  
 counter = 1  
 candidate = path  
 while os.path.exists(candidate):  
 candidate = f"{base}({counter}){ext}"  
 counter += 1  
 return candidate  
  
 def convert\_font(self):  
 font\_path = self.entry.get()  
 if not os.path.isfile(font\_path):  
 messagebox.showerror("Error", "Please select a valid TTF font file.")  
 return  
  
 try:  
 self.status.config(text="Processing... Please wait.")  
 self.root.update()  
  
 output\_path = self.generate\_lvgl\_font\_c(font\_path)  
  
 self.status.config(text=f"Success! Generated file: {output\_path}")  
 messagebox.showinfo("Done", f"LVGL font file generated:\n{output\_path}")  
 except Exception as e:  
 messagebox.showerror("Error", str(e))  
 self.status.config(text="")  
  
 def format\_bitmap\_bytes(self, buffer):  
 lines = []  
 line = " "  
 for i, byte in enumerate(buffer):  
 line += f"0x{byte:02X}, "  
 if (i + 1) % 12 == 0:  
 lines.append(line)  
 line = " "  
 if line.strip():  
 lines.append(line)  
 return "\n".join(lines)  
  
 def write\_glyph\_bitmap\_comment(self, f, char\_code):  
 ch = chr(char\_code)  
 if ch == '"':  
 ch = '\\"'  
 elif ch == '\\':  
 ch = '\\\\'  
 f.write(f"\n/\* U+{char\_code:04X} \"{ch}\" \*/\n\n")  
  
 def generate\_lvgl\_font\_c(self, font\_path):  
 FONT\_SIZE = 16  
 CHAR\_START = 0x20  
 CHAR\_END = 0x7F  
  
 face = freetype.Face(font\_path)  
 face.set\_pixel\_sizes(0, FONT\_SIZE)  
  
 # Initial output file name  
 base\_name = os.path.splitext(os.path.basename(font\_path))[0]  
 initial\_path = os.path.join(os.path.dirname(font\_path), f"{base\_name}.c")  
  
 # Get unique output filename to avoid overwrite  
 output\_filename = self.get\_unique\_filename(initial\_path)  
  
 glyph\_bitmap = []  
 glyph\_dsc = []  
 bitmap\_index = 0  
  
 with open(output\_filename, "w", encoding="utf-8") as f:  
 # Write header  
 f.write(f"/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n")  
 f.write(f"\* Size: {FONT\_SIZE} px\n\n")  
 f.write(f"\* Bpp: 1\n\n")  
 f.write(f"\* Opts: --font {font\_path} --size {FONT\_SIZE} --bpp 1 --no-compress --format lvgl\n\n")  
 f.write(f"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/\n\n")  
  
 f.write("#ifdef LV\_LVGL\_H\_INCLUDE\_SIMPLE\n\n#include \"lvgl.h\"\n\n#else\n\n#include \"lvgl/lvgl.h\"\n\n#endif\n\n")  
  
 macro\_name = base\_name.upper().replace(" ", "\_").replace("-", "\_")  
 f.write(f"#ifndef {macro\_name}\n#define {macro\_name} 1\n\n#endif\n\n")  
  
 # Bitmaps  
 f.write("/\*-----------------\n\n\* BITMAPS\n\n\*----------------\*/\n\n")  
 f.write("/\*Store the image of the glyphs\*/\n\n")  
 f.write("static LV\_ATTRIBUTE\_LARGE\_CONST const uint8\_t glyph\_bitmap[] = {\n")  
  
 for code in range(CHAR\_START, CHAR\_END + 1):  
 try:  
 face.load\_char(chr(code), freetype.FT\_LOAD\_RENDER | freetype.FT\_LOAD\_TARGET\_MONO)  
 except:  
 continue  
  
 bmp = face.glyph.bitmap  
 width = bmp.width  
 rows = bmp.rows  
 buffer = bmp.buffer  
  
 self.write\_glyph\_bitmap\_comment(f, code)  
  
 row\_bytes = (width + 7) // 8 if width != 0 else 0  
  
 for y in range(rows):  
 f.write(" ")  
 for b in range(row\_bytes):  
 idx = y \* row\_bytes + b  
 if idx < len(buffer):  
 f.write(f"0x{buffer[idx]:02X}")  
 else:  
 f.write("0x00")  
 if not (y == rows -1 and b == row\_bytes -1):  
 f.write(", ")  
 f.write(",\n")  
  
 adv = face.glyph.advance.x >> 6  
 left = face.glyph.bitmap\_left  
 top = face.glyph.bitmap\_top  
  
 glyph\_dsc.append({  
 "bitmap\_index": bitmap\_index,  
 "adv\_w": adv \* 16,  
 "box\_w": width,  
 "box\_h": rows,  
 "ofs\_x": left,  
 "ofs\_y": top,  
 })  
  
 bitmap\_index += len(buffer)  
  
 f.write("};\n\n")  
  
 # Glyph Description  
 f.write("/\*-----------------\n\n\* GLYPH DESCRIPTION\n\n\*----------------\*/\n\n")  
 f.write("static const lv\_font\_fmt\_txt\_glyph\_dsc\_t glyph\_dsc[] = {\n")  
 f.write(" {.bitmap\_index = 0, .adv\_w = 0, .box\_w = 0, .box\_h = 0, .ofs\_x = 0, .ofs\_y = 0},\n")  
  
 for d in glyph\_dsc:  
 f.write(f" {{.bitmap\_index = {d['bitmap\_index']}, .adv\_w = {d['adv\_w']}, .box\_w = {d['box\_w']}, .box\_h = {d['box\_h']}, .ofs\_x = {d['ofs\_x']}, .ofs\_y = {d['ofs\_y']}}},\n")  
  
 f.write("};\n\n")  
  
 # Character Mapping  
 f.write("/\*-----------------\n\n\* CHARACTER MAPPING\n\n\*----------------\*/\n\n")  
 f.write("static const lv\_font\_fmt\_txt\_cmap\_t cmaps[] = {\n")  
 f.write(" {\n")  
 f.write(f" .range\_start = {CHAR\_START},\n")  
 f.write(f" .range\_length = {CHAR\_END - CHAR\_START + 1},\n")  
 f.write(" .glyph\_id\_start = 1,\n")  
 f.write(" .unicode\_list = NULL,\n")  
 f.write(" .glyph\_id\_ofs\_list = NULL,\n")  
 f.write(" .list\_length = 0,\n")  
 f.write(" .type = LV\_FONT\_FMT\_TXT\_CMAP\_FORMAT0\_TINY\n")  
 f.write(" }\n")  
 f.write("};\n\n")  
  
 # Font Descriptor  
 f.write("/\*-----------------\n\n\* FONT DESCRIPTION\n\n\*----------------\*/\n\n")  
 f.write("static lv\_font\_fmt\_txt\_dsc\_t font\_dsc = {\n")  
 f.write(" .glyph\_bitmap = glyph\_bitmap,\n")  
 f.write(" .glyph\_dsc = glyph\_dsc,\n")  
 f.write(" .cmaps = cmaps,\n")  
 f.write(f" .cmap\_num = 1,\n")  
 f.write(" .bpp = 1,\n")  
 f.write(" .kern\_dsc = NULL,\n")  
 f.write(" .kern\_scale = 0,\n")  
 f.write(" .kern\_classes = 0,\n")  
 f.write(" .bitmap\_format = 0\n")  
 f.write("};\n\n")  
  
 # Public Font  
 f.write("/\*-----------------\n\n\* PUBLIC FONT\n\n\*----------------\*/\n\n")  
 f.write("lv\_font\_t lv\_font\_custom = {\n")  
 f.write(" .get\_glyph\_dsc = lv\_font\_get\_glyph\_dsc\_fmt\_txt,\n")  
 f.write(" .get\_glyph\_bitmap = lv\_font\_get\_bitmap\_fmt\_txt,\n")  
 f.write(f" .line\_height = {FONT\_SIZE},\n")  
 f.write(" .base\_line = 0,\n")  
 f.write(" .dsc = &font\_dsc\n")  
 f.write("};\n\n")  
  
 f.write(f"#endif /\* {macro\_name} \*/\n")  
  
 return output\_filename  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 app = LVGLFontConverterApp(root)  
 root.mainloop()

**Steps to make it .exe application and user interact**  
  
create a standalone interactive .exe application from your Python GUI code (e.g., font2c\_lvgl.py) that will be generated in the dist folder, follow these steps:

1. **Ensure your Python GUI script is ready**, e.g., the full code for font conversion with tkinter interface as discussed.
2. **Install PyInstaller** if not already installed:

text

pip install pyinstaller

1. **Open a terminal or command prompt** and navigate to your script's directory.
2. **Run PyInstaller with the following command:**

text

pyinstaller --onefile --windowed font2c\_lvgl.py

* + --onefile packs all dependencies into a single executable file.
  + --windowed suppresses the console window for GUI applications.

1. **After completion, locate the .exe file in the dist folder** inside your project directory.
2. **Run the .exe application** — an interactive GUI window will appear where the user can select a .ttf file and generate the LVGL .c font file.

**Separate Python Script to Delete Old .exe and Build New .exe**

Save this small script as build\_clean.py in your project directory:  
  
Python code for delete old exe in dist folder and create new one

python

**import** os

**import** subprocess

exe\_path = os.path.join("dist", "font2c\_lvgl.exe")

**if** os.path.exists(exe\_path):

**print**(f"Deleting existing executable: {exe\_path}")

os.remove(exe\_path)

**print**("Building executable...")

subprocess.run(["pyinstaller", "--onefile", "--windowed", "font2c\_lvgl.py"], check=True)

**print**("Build completed.")

**How to use**

1. Implement the first Python script as font2c\_lvgl.py.
2. Run this for font conversion in GUI.
3. To build or rebuild .exe use:

text

python build\_clean.py

This deletes the old dist/font2c\_lvgl.exe if it exists and rebuilds your executable with PyInstaller.

This way you have clean separation:

* GUI font converter app for end user functionality
* Build script for developer convenience automating clean builds

Steps to make it desktop application  
  
Here is a detailed step-by-step guide including adding a custom icon, to create a desktop installable application from your Python GUI script:

**Step 1: Prepare Your Python GUI Script**

* Ensure font2c\_lvgl.py is working and tested with your font conversion tkinter GUI.

**Step 2: Prepare a .ico icon file**

* Create or obtain an .ico file (e.g., app\_icon.ico) for your application’s icon.
* Use an online converter or image editor to convert PNG/JPG to ICO format if needed.

**Step 3: Install PyInstaller**

* Open command prompt or terminal
* Run:

text

pip install pyinstaller

**Step 4: Delete Old Executable (Optional but Recommended)**

* To prevent conflicts, manually delete or automate deletion of existing .exe in the dist folder.

**Step 5: Build the Executable with Icon**

* Run PyInstaller with icon flag:

text

pyinstaller --onefile --windowed --icon=LVGL\_FontGen.ico font2c\_lvgl.py

* --onefile packages everything into a single exe
* --windowed hides console with GUI apps
* --icon=app\_icon.ico embeds your icon

**Step 6: Locate the Executable**

* After the build, find your .exe file here:

text

dist\font2c\_lvgl.exe

* This exe will have the custom icon embedded.

**Step 7: Create Installer (Optional)**

* Download and install **Inno Setup**: <https://jrsoftware.org/isdl.php>
* Write installer script (.iss) that packages your .exe and installs it with Start Menu and Desktop shortcuts.
* Compile installer with Inno Setup.

**Step 8: Test Installation and Run**

* Run your installer on a clean machine or VM.
* Verify desktop, start menu icons appear with your custom icon.
* Run the app and ensure GUI font conversion works as expected.

**Summary**

| **Step** | **Action** |
| --- | --- |
| 1 | Prepare Python GUI script |
| 2 | Prepare .ico icon file |
| 3 | Install PyInstaller |
| 4 | (Optional) Remove pre-existing exe |
| 5 | Build exe with icon using PyInstaller |
| 6 | Locate and test the exe |
| 7 | (Optional) Create installer using Inno Setup |
| 8 | Install and test |

iss file script - setup.iss  
  
[Setup]

AppName=LVGL Font Generator

AppVersion=1.0

DefaultDirName={pf}\LVGLFontGenerator

DefaultGroupName=LVGL Font Generator

OutputBaseFilename=LVGLFontGenerator\_Installer

Compression=lzma

SolidCompression=yes

WizardStyle=modern

SetupIconFile=LVGL\_FontGen.ico

[Languages]

Name: "english"; MessagesFile: "compiler:Default.isl"

[Files]

Source: "dist\font2c\_lvgl.exe"; DestDir: "{app}"; Flags: ignoreversion

[Icons]

Name: "{group}\LVGL Font Generator"; Filename: "{app}\font2c\_lvgl.exe"; IconFilename: "{app}\LVGL\_FontGen.ico"

Name: "{commondesktop}\LVGL Font Generator"; Filename: "{app}\font2c\_lvgl.exe"; IconFilename: "{app}\LVGL\_FontGen.ico"; Tasks: desktopicon

[Tasks]

Name: "desktopicon"; Description: "Create a &Desktop icon"; GroupDescription: "Additional icons:"

[Run]

Filename: "{app}\font2c\_lvgl.exe"; Description: "Launch LVGL Font Generator"; Flags: nowait postinstall skipifsilent  
  
  
Python code