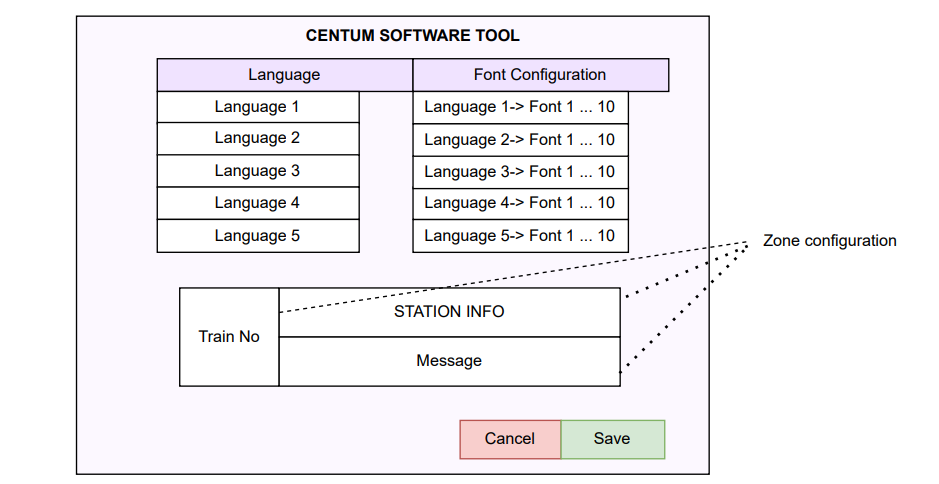
## Centum Configuration Tool – Font Converter

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.

LED-Font-Converter: Comprehensive Step-by-Step Task Details

**1. Environment Setup and Prerequisites**

**Input:**

* Developer machine (Windows preferred for building).
* Internet access to install Python and dependencies.

**Process:**

* Install **Python 3.7+** (ensure added to PATH).
* Install Python libraries via pip: freetype-py, PyQt5, Pillow, PyInstaller, setuptools, and wheel.
* Install **Node.js v14+** if planning to use LVGL's font conversion CLI tool.
* Install **Inno Setup Compiler (ISCC.exe)** for building Windows installer (download from official source).

**Requirements:**

* Python development environment (3.7+).
* Pip package manager.
* Admin rights may be required for installations.

**How to start:**

* Verify Python installation with python --version.
* Install packages with:

text

pip install freetype-py pyqt5 pillow pyinstaller setuptools wheel

* Download and install Node.js and ISCC.exe separately.

**Tools used:**

* Python 3.7+
* Pip
* PyInstaller
* Node.js (optional)
* Inno Setup Compiler

**2. Application Initialization**

**Input:**

* Source Python scripts (main GUI app, font conversion modules).

**Process:**

* Run the main Python GUI script (e.g., main.py or similarly named file in source).
* GUI initializes multiple font converter widgets.
* Application loads configuration (if any).

**Requirements:**

* All dependency libraries installed.

**How it works:**

* main.py imports GUI modules and sets up PyQt5 application window.
* Multiple widgets created dynamically to allow batch font conversion.

**Script internals:**

* Main script entry point: if \_\_name\_\_ == '\_\_main\_\_': block creates QApplication and main window.
* Widget classes handle user input elements.

**Output:**

* Running application GUI window ready for user interaction.

**How to start:**

* Navigate to source folder.
* Run:

text

python main.py

**3. Font Selection and Configuration**

**Input:**

* User interaction: Selecting font name, TTF file, font size, bpp, Unicode range.

**Process:**

* File dialog opens for TTF selection.
* User enters hex Unicode range (start and end).
* User sets font size (pixels) and bits per pixel (quality).

**Requirements:**

* Accessible TTF font files on disk.

**How it works:**

* GUI fields validate user inputs.
* Selected font path and parameters stored in widget instance variables.

**Script functions called:**

* File dialog function (e.g., QFileDialog.getOpenFileName).
* Input validators applied to fields.

**Output:**

* Widget ready with user-specified font settings for conversion.

**Tools used:**

* PyQt5 widgets for input.

**How to start:**

* Click buttons or input fields in GUI to provide required font data.

**4. Font Rendering and Bitmap Generation**

**Input:**

* Selected TTF file path.
* Font size, bpp, Unicode range.

**Process:**

* The application loads font through freetype-py.
* Loops through Unicode codepoints specified.
* For each character:
  + Render glyph bitmap at given size and bpp.
  + Extract bitmap pixel data.

**Requirements:**

* Working freetype-py bindings.

**How it works:**

* freetype.Face initialized with font file.
* face.set\_char\_size called with font size \* 64 (FreeType uses 1/64th pixels).
* face.load\_char() loads glyph.
* Glyph bitmap buffer accessed.

**Script functions called:**

* freetype-py methods: Face(), set\_char\_size(), load\_char().
* Bitmap data processing functions.

**Output:**

* Raw bitmap pixel arrays for all glyphs in range.

**How to start:**

* Trigger conversion button in GUI after inputs are valid.

**5. C Source Code Generation for LVGL**

**Input:**

* Glyph bitmap arrays.
* Font metadata (size, bpp, unicode map).

**Process:**

* Formats glyphs and metadata into C data structures:
  + Bitmap arrays as static const uint8\_t bitmap[]
  + Glyph descriptors with dimensions and offsets
  + Unicode character mapping arrays
* Wraps all in structures matching LVGL font API.

**Requirements:**

* Understanding of LVGL font file format.

**How it works:**

* Code templates populated with bitmap and descriptor data.
* Strings concatenated to form full C source.

**Script functions called:**

* Formatting helpers for arrays and structs.
* File write functions.

**Output:**

* .c source file containing font data

**File naming:**

* Auto-renamed to avoid duplication, saved next to original TTF.

**How to start:**

* Conversion script auto-generates after bitmap extraction.

**6. Saving Output and File Management**

**Input:**

* Generated C source code string.
* Output directory (same as TTF).

**Process:**

* Check if output filename exists.
* If exists, append incremental suffix.
* Write the .c font file to disk.

**Requirements:**

* File system write permission.

**How it works:**

* Uses Python os and pathlib modules to check paths and rename.
* Standard file open/write used.

**Output:**

* Created .c file accessible for embedded development.

**How to start:**

* Auto-saves after conversion completes.

**7. Build Automation Script: build\_and\_installer.py**

**Input:**

* Source Python scripts
* Icon files (\*.ico)
* Inno Setup script (.iss)

**Process:**

* Checks PyInstaller installed; installs if missing.
* Runs PyInstaller to create single executable for Windows:
  + Uses --onefile and --windowed flags.
  + Applies custom icon.
* Cleans previous builds before packaging.
* Invokes Inno Setup compiler (ISCC.exe)
  + Compiles .iss script to create Windows installer executable.
* Removes old installation directories before building new installer.

**Requirements:**

* Python environment with PyInstaller.
* ISCC.exe must be installed and accessible.
* Icon file present for customization.

**How it works:**

* Automates repetitive build tasks
* Encapsulates complex CLI commands in Python subprocess calls

**Script Details:**

* Uses subprocess.run() for command-line tools.
* Paths and filenames configured in variables or config.

**Output:**

* Standalone .exe application file
* .exe installer for Windows with installation and uninstall support

**How to start:**

* Run:

text

python build\_and\_installer.py

**8. Installation and Deployment**

**Input:**

* Generated installer .exe

**Process:**

* User runs installer on Windows PC.
* Windows installs application under chosen directory.
* Adds uninstall entry to system.

**Requirements:**

* Windows OS

**How it works:**

* Inno Setup installer manages file copying and shortcuts.
* Removes previous versions as configured.

**Output:**

* Installed desktop app ready to run.

**How to start:**

* Double-click installer and follow wizard.

This describes end-to-end operation of LED-Font-Converter from environment setup, through GUI usage to conversion internals, all the way to building and distributing the Windows desktop app.