WiPhone Programming Manual

Revision history

Ver.	Date	Author	Comment
0.6	10 Jun 2019	Andriy Makukha	Instructions for early evaluators
0.7.1	7 Oct 2019	Andrii Makukha	Step-by-step guide for developing custom apps

Prerequisites

Hardware

For compiling and editing WiPhone firmware, you will need a computer capable of running Arduino Desktop IDE (with either Linux, MacOS or Windows operating systems) and a microUSB cabel.

Software

The following software packages are required to compile WiPhone firmware:

1. Arduino Desktop IDE
   https://www.arduino.cc
2. Arduino core for ESP32 (Arduino-ESP32)
   https://github.com/espressif/arduino-esp32
3. Arduino plugin for uploading files to ESP32 file system
   https://github.com/me-no-dev/arduino-esp32fs-plugin

If you want to contribute to the firmware, you should also have the Git source-control management tool: https://git-scm.com/

Installing Arduino Desktop IDE

Download an installer for your platform from the official Arduino website:
https://www.arduino.cc
Download page: https://www.arduino.cc/en/Main/Software

Follow the installation procedure for each operating system:

- For Linux, download the archive, extract it, run file “install.sh”.
  See official instructions for more details: https://www.arduino.cc/en/Guide/Linux

- For MacOS, copy the file from downloaded archive into the Applications folder.
  See official instructions for more details:
  https://www.arduino.cc/en/Guide/MacOSX
- For Windows, download and run the installer.
  See official instructions for more details:
  https://www.arduino.cc/en/Guide/Windows

Installing Arduino-ESP32

Follow the official installation instructions: https://github.com/espressif/arduino-esp32#installation-instructions

If your Arduino IDE is recent enough (ver. 1.8+), you can install Arduino-ESP32 with Arduino IDE's Boards Manager:

1. Start Arduino Desktop IDE.
2. Open Preferences window.
3. Enter "https://dl.espressif.com/dl/package_esp32_index.json" URL (without quotes) into the *Additional Board Manager URLs* field.
4. Open the *Boards Manager...* window by navigating to *Tools* > *Board: "..."* > *Boards Manager...*
5. Search for "esp32" and install esp32 package by Espressif Systems.

Choosing a board type

After installing Arduino-ESP32 (or, equivalently, esp32 package in the Boards Manager), select a WiPhone-compatible board with 16 MB flash by navigating to *Tools* > *Board: "..."* menu. Choosing "M5Stack-FIRE" is currently recommended!

NOTE: Choosing your board will also afftect the partitionning of the internal flash (this setting can be changed or viewed by *Tools* > *Partition scheme*).

WARNING: Changing partition scheme might overwrite the internal flash file system (SPIFFS) causing irreversible data loss of WiPhone data files (like phonebook and SIP accounts). Stick to a single board type / partition scheme early in the development to avoid data loss.

ADVANCED USER NOTE: Avaliable ESP32 boards and partition schemes are specified here: https://github.com/espressif/arduino-esp32/blob/master/boards.txt

Installing Arduino plugin for uploading files

This plugin is required to upload data files to WiPhone's internal flash (SPIFFS). It is needed, for example, to load the ringtone file and configuration. It also formats the SPIFFS partition of ESP32 to allow storing WiPhone data (like phonebook and SIP accounts) into the internal flash.

WARNING: Loading files with this plugin into WiPhone will overwrite any existing files in the SPIFFS partition, causing irreversible data loss. Use this only if you know what you are doing!

Installation procedure:

1. Download the plugin archive: https://github.com/me-no-dev/arduino-esp32fs-plugin/releases/download/1.0/ESP32FS-1.0.zip
2. Extract the archive and copy the extracted directory ESP32FS into the tools subdirectory of Arduino sketchbook directory:
   - On Linux, create directory ~/Arduino/tools/ (you can run "mkdir -p ~/Arduino/tools/" in terminal) and copy the ESP32FS direcory into it.
   - On MacOS, create directory ~/Documents/Arduino/tools/ (you can run "mkdir -p ~/Documents/Arduino/tools/" in terminal) and copy the ESP32FS direcory into it.

Consult the official installation instructions as well: https://github.com/me-no-dev/arduino-esp32fs-plugin#Installation

Obtaining a copy of firmware

In the future we will publish the firmware on GitHub.
For now (before the rewards are delivered), we will send copies of the firmware to crowdfunding backers who sign an NDA. If you want a copy, please send an email to Ben (ben@wiphone.io).

Compiling and uploading firmware

After obtaining firmware, make sure that all of the files and subdirectories are stored in a directory called “WiPhone”. Then, to compile and upload the firmware to WiPhone:

1. Connect WiPhone to your computer with a microUSB cable (the cable should have a microUSB plug on one end to connect to WiPhone, and any other connector that is compatible with your comnputer’s sockets on another end, like USB-A or USB-C plug).
2. Open the file WiPhone.ini from the project directory in Arduino IDE.
3. Press button “Upload” in the top left corner of the Arduino IDE window (the button looks like a right arrow).

Uploading data files

WiPhone allows data files to be stored permanently in the internal Flash file system (SPIFFS). Particularly, this partition is used to store the ringtone music file and configuration data.

To upload data files from subdirectory “data“ to SPIFFS, do the following:

1. Make sure that the plugin for uploading files to ESP32 is installed (otherwise, see the dedicated section above).
2. Navigate to “Tools” menu in Arduino IDE and press “ESP32 Sketch Data Upload”.

What is what in the project directory

Here are some files and subdirectories from the source directory:

- **WiPhone.ino**
  - Arduino project file: a C++ file with the main loop and the main phone logic
- **GUI.h**
- **GUI.cpp**
  - main GUI logic files (widgets and apps are defined here)
- **tinySIP.h**
- **tinySIP.cpp**
  - our implementation of the SIP protocol
- **src/**
  - directory for code that is not in active development (these files are crucial for WiPhone's operation, but will not be opened by Arduino IDE by default, unlike files in the main directory)
  - **src/TFT_eSPI/**
    - a library for ESP32 to drive the screen at fast speed; includes sprites and font rendering capabilities
  - **src/VoIP/**
    - audio codecs and other definitions for Voice-over-IP
  - **src/assets/**
    - static data components (such as fonts, icons, background image, etc.)
  - **src/drivers**
    - code that operates different integrated circuits (ICs) of the WiPhone
- **data/**
  - files to be loaded into WiPhone's internal flash file system (SPIFFS)
- **tools/**
  - custom Python scripts to generate "assets", namely 3-bit antialiased fonts and icons
- other WiPhone source files in the main directory

Developing custom “apps”

Developing custom apps in WiPhone’s Arduino firmware is fairly easy and straightforward. Even a beginner programmer with knowledge of C++ should be able to figure it out after learning how existing apps work. Having said that, there are some non-transparent parts that will be covered and discussed in this chapter. As well as some principal design disadvantages that will limit the way apps can be developed (see the disclaimer below).

Disclaimer: single thread

As of September 2019 the WiPhone firmware uses cooperative multitasking. We are not using FreeRTOS threads because we encountered instability with the network stack in our early trials. Therefore any custom apps that are developed for WiPhone must follow a cooperative multi-tasking approach: never lock or take more than 10-100 ms of CPU time per call. Failing to do so will make the interface responsive. For example, the entire phone will appear to be frozen if a non-asynchronous network connection is attempted but cannot be established rapidly or at all. This area is open for improvement later, but for now we are concentrating on bringing the overall phone functionality up to the feature level we feel is minimally acceptable before coming back to add threads. If someone else wants to implement a patch please contact us so we can discuss feasibility.

Steps for adding a custom app

1. Declare a unique ID for the new app. For this purpose, add a new element to the ActionID enumeration:

Find “typedef enum ActionID” in file “GUI.h”. Add a unique name below the line “GUI_BASE_APP = 0x4000”. For example, “GUI_APP_NEW_CUSTOM”.

2. Declare the app class. The easiest way to do this is to find an existing app similar to the one you want to build, copy its code, change class name and then change that code to do the things you want.

The app class declaration goes into “GUI.h” file, definitions of its methods go into “GUI.cpp”.

The app class must derive from “WiPhoneApp” or of its derivatives (like “WindowedApp” or “FocusableApp”, or both).

Make sure the method “getId” returns the ID that you created in the first step.

3. Define the app methods. Steps 2 and 3 are the actual app implementation. This is usually the biggest and longest step.

This is done in file “GUI.cpp”.

Each app defines two main methods: “processEvent()” and “redrawScreen()”.

“processEvent(EventType event)” is the method that is called by GUI to allow your app process events and update its internal state accordingly. The events are, for example, button presses, timers (as requested by your app) or scheduled events (as scheduled by your app).

“redrawScreen(bool redrawAll)” is the method that is called by GUI to allow your app to redraw the screen partially and/or telling your app that it should redraw the screen entirely (like after the screen was redrawn by some other app, like a call or a screen lock).

4. Add your app ID into the main menu.

For that purpose, find “GUIMenuItem menu” in file “GUI.h”.

Increase size of the array by 1.

Add a line of the form “{ XX, YY, “My custom app", "", "", GUI_APP_NEW_CUSTOM },“ into the definition of the array “menu”; XX – should be a unique ID of the menu item (for simplicity, it should be the current size of the array), YY – is parent ID, or ID of the submenu, in which you want your app to appear.

5. Instantiate your app. This is, basically, just letting the GUI know how to call your app’s constructor.

Find definition of method “GUI::enterApp()” in file “GUI.cpp” and add code to create a new object out of your app class. (Do it similarly to other apps.)