# uBTIX Settings Editor User Manual

A Guide to getting the most out of your uBITX running the KD8CEC Software

Mark Hatch (AJ6CU)
March 4, 2023

# Preface

I had not originally planned to write the uBITX Settings Editor. My big plan was to first port the KD8CEC software from its lowly home on the Arduino Nano to faster and more modern processors. My hope was that by breaking free of the constraints of the Nano, I would open the door to not only a much needed restructuring of the code, but also make it possible for more contributors to extend its functionality. And just perhaps, this software could turn into a widely used, open source ham radio code that might also embrace future uBITX architectures or even radios designed and build by others.

Well, that was a good goal…

Unfortunately, as I was in the middle of porting the KD8CEC software to the Arduino BLE (all ready had it working on the Arduino IOT), I discovered that the original uBITX Memory Manager refused to talk to the uBITX with the BLE. Initially, I assume it was my issue and kept going. But then I found that the uBITX Memory Manager didn't like the Arduino RP Connect (a RaspberryPi Pico in an Arduino package) either. But then I found it liked the Teensy. So what was going on here?

The problem was that Dr. Lee never released the source code to the uBITX Memory Manager. So I couldn't debug the problem. And if I didn't have a working uBITX Memory Manager, then no one would want to use KD8CEC running on a Raspberry Pi Pico or any other processor. And there was a real possibility that as the KD8CEC software evolved the two pieces of software would increasingly become unwilling to communicate.

That left me no choice but to rewrite the uBITX Memory Manager if I wanted to proceed with the big plans for the KD8CEC software.

Initially, I tried to constrain the problem. Within about a month of work, I had two programs that could read/write to the EEProm with a human readable and editable XML file as the save file format. Unfortunately, after making some stupid typo errors while I was editing my own XML files, I came to the conclusion that this was not a satisfactory solution.

So the scope of the problem increased to embrace a more human friendly application with a graphical user interface. This actually increased the scope by multiple factors. Not only is designing a GUI hard (both in design and coding), it also means understanding what was really going on at a much deeper level. The lack of source to the uBITX Memory Editor made this task harder as I had to black box it and change inputs, see what the outputs were generated, and then go look in the KD8CEC code to see what really happened.

So 5 months later…

I have finished the Beta version of the new uBITX Settings Editor. I believe it is a great leap forward. Not only is it open source, a better organized GUI, but it is also available on MacOS and Linux in addition to the original Windows platform hosted the uBTIX Memory Manager.

I hope you like it too and I look forward to your feedback!

73

Mark (AJ6CU), March 4, 2023

# Introduction

The heard of the uBITX transceiver is a microcontroller (MCU) . The uBITX is manufactured by HF Signals (https://www.hfsignals.com/) using an Arduino Nano V3 MCU. The MCU includes flash memory of approximately 32kb as well as an 1024 bytes of Electrically Erasable Programmable Read-Only Memory (EEPROM). The flash memory holds the firmware or program where as the EEPROM is used to store settings (e.g. calibration information, last used frequencies and modes, tuning rates, etc.).

Although functional, the original firmware provided by HF Signals only supported a 2 line character LCD As a result, Dr. Ian Lee (KD8CEC) developed an enhanced version of the firmware (we will refer to it as KD8CEC throughout this manual) that provided a graphical user interface that used the screens from Nextion. His efforts are documented in a blog format at www.hamskey.com.

The "eye candy" and enhanced functionality of the KD8CEC software is what first catches the attention of most hams. However, what many miss is that the KD8CEC software is of limited value without a tool to easily tailor it to the users needs. Sure you can set the CW keyer speed through the GUI (you will sometimes equivalently referring to this as the "UX" which stands for User Experience), but suppose you need to:

- tune when your dot or dash is recognized as a dot or dash respectively.
- to use the automated keyer that is part of KD8CEC and you want to define the "canned message" (i.e., CQ CQ DE…).
- use the uBITX as a WSPR beacon.
- to tell your external linear amp what band you are sending on so it can select the right bandpass filter.

Or you get the idea… You can't practically do this at the Radio. That is where you need a tool to tailor the setings of the uBITX to meet your needs. Originally, Dr. Lee wrote the uBITX Memory Manager (available at: https://github.com/phdlee/ubitx ) and kept it updated as he added new functionality. However, for reasons that I discussed in the Preface, that code has "virtually rotted" and it was time for a rewrite.

This document is the user manual for my rewrite of this software. I have attached the name "uBITX Settings Editor' (lets refer to it as the "SE" from now on) to it because that seemed to provide a clearer communication on its purpose.

# The Plan for our Journey

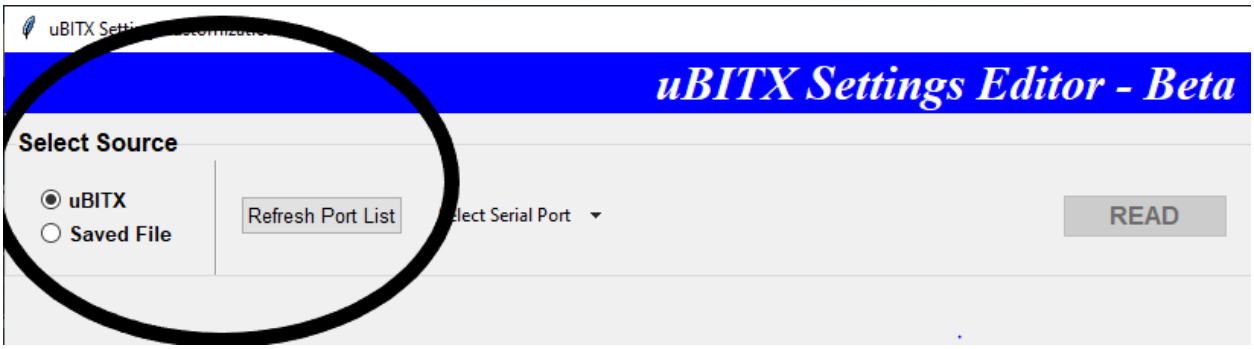
The next couple sections will give you a basic orientation to the software and suggest a process on how you should use it. The software makes heavy usage of tooltips and provides larger hints adjacent to key functionality. I suspect that after this basic overview, you will be able to jump in and just use the software.

The rest of the sections will focus on each basic area of functionality and do a "deep dive" where I will try to share all my knowledge that I gained during the reverse engineering process. Not guaranteeing that this knowledge is relevant, interesting, or even accurate. But I am hoping that by sharing, that the knowledge is preserved.

## Software at 20,000 ft (6096 meters)

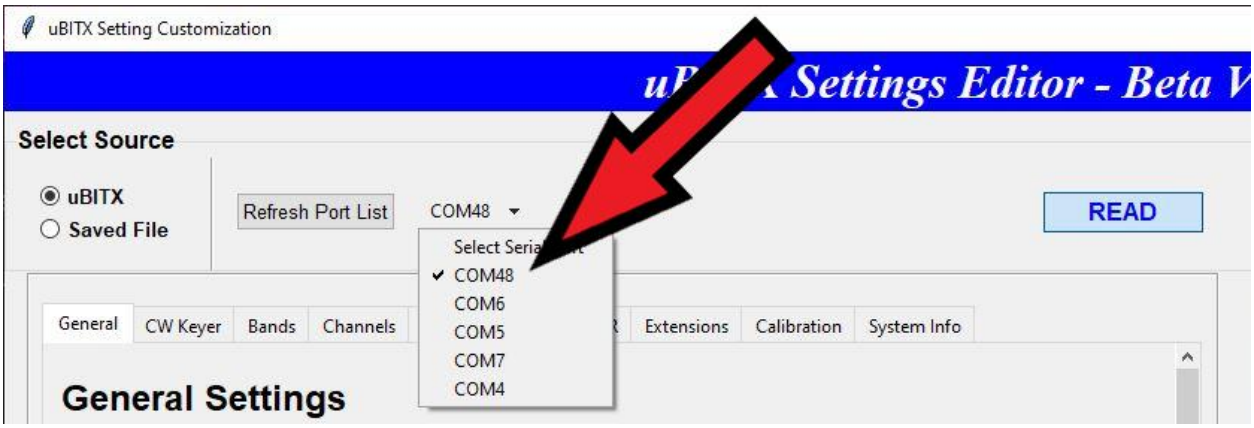
After you start the SE, you will be greeted by a screen that looks like:


A lot of empty space, right? Lets look at the top left section.




This is where you select the source for all your uBITX settings. The most common choice is that you will read it from the uBITX itself. And that is the default as the uBITX “button” is selected.  Assuming that is

what you want to do, then you need to select the com port that your radio is sitting on like in the following screenshot:

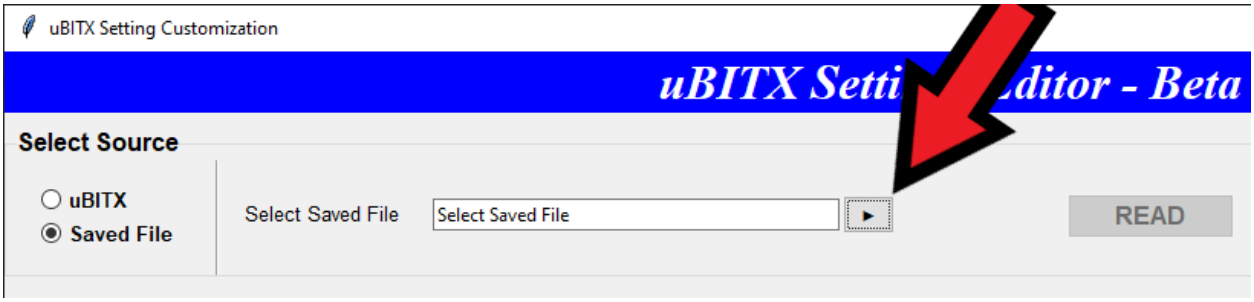



Obviously, the ugly red arrow with black outline points out that the com port listing box was clicked and then COM48 (in my case) was selected. Once selected the READ button on the far right becomes active and you can read the settings.

I suspect the hardest part of this step is not UX on how you select the com port, but trying to figure out **which com port** corresponds to your radio. On Windows, the old trick is to start the "Device Manager", go to the Com port section and plug in your radio. And the new one is the winner. I suspect there are similar tricks for the other platforms (including guessing and when guessing right, writing it down for the future), but finding these tricks is left as an exercise for you.

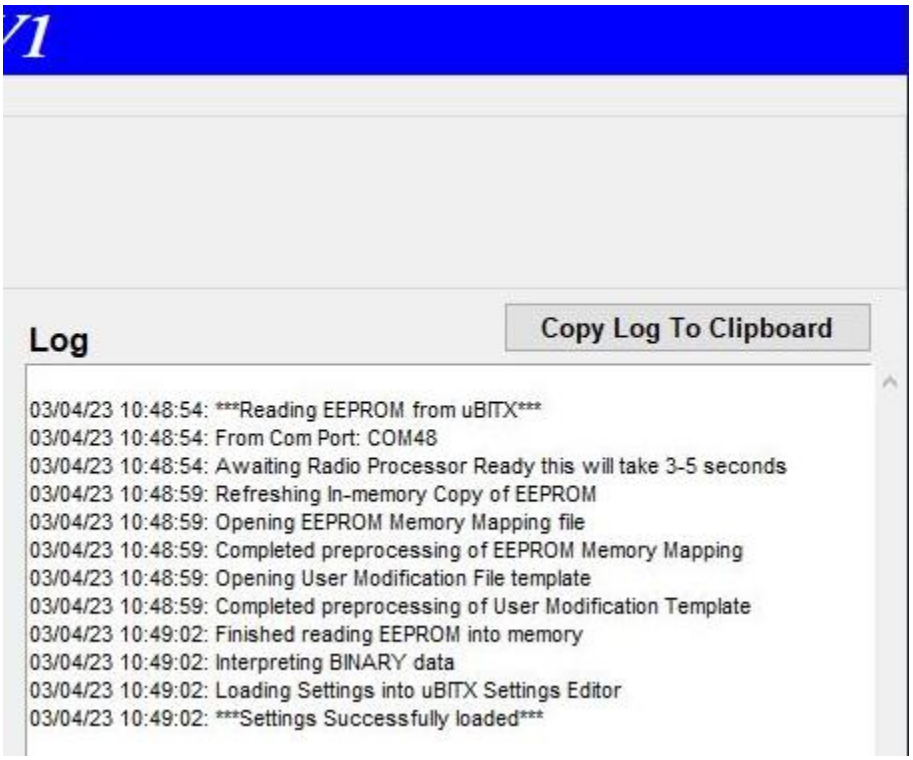
So why do you want to read your input from a file? Well, you really should backup your uBITX settings right? And perhaps you switch processors in the future and want to restore your settings? Or maybe even your processor dies. You will never regret having a backup, but not having one when you need it can be disappointing.

The screen below shows what happens when you select to read the settings from a file. The com port selection portion of the application is replaced by a file selector box. Clicking on that small arrow that the large red arrow points at brings up a standard file selector box that you can use to locate your backup file.




So again, like in the com port section, selecting a file will allow you to click the READ button on the right.

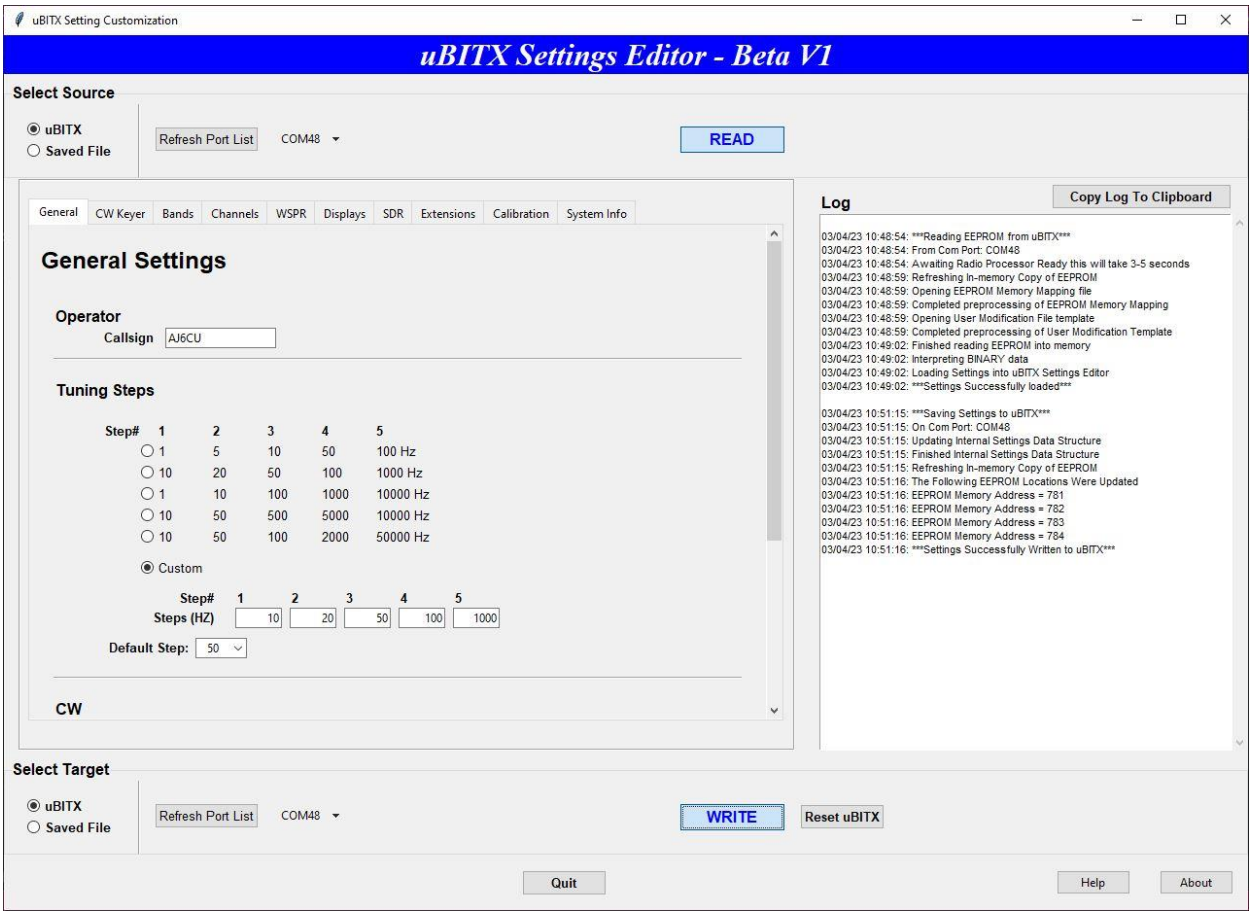
As the SE reads in your settings, you will start to see action in the area called “Log” that is on your right. For example, the following Log snipit was the result of reading the settings from a uBITX:




The log is mostly useful when you have done something wrong. In this case, the second line of the log identifies the Com port where radio was connected (perhaps useful for the future). The rest of the log just talks about what is going on. It can be useful as a “heartbeat” for the application. Probably in the future, I will either reduce the logging or provide a way to turn verbose logging on and off.

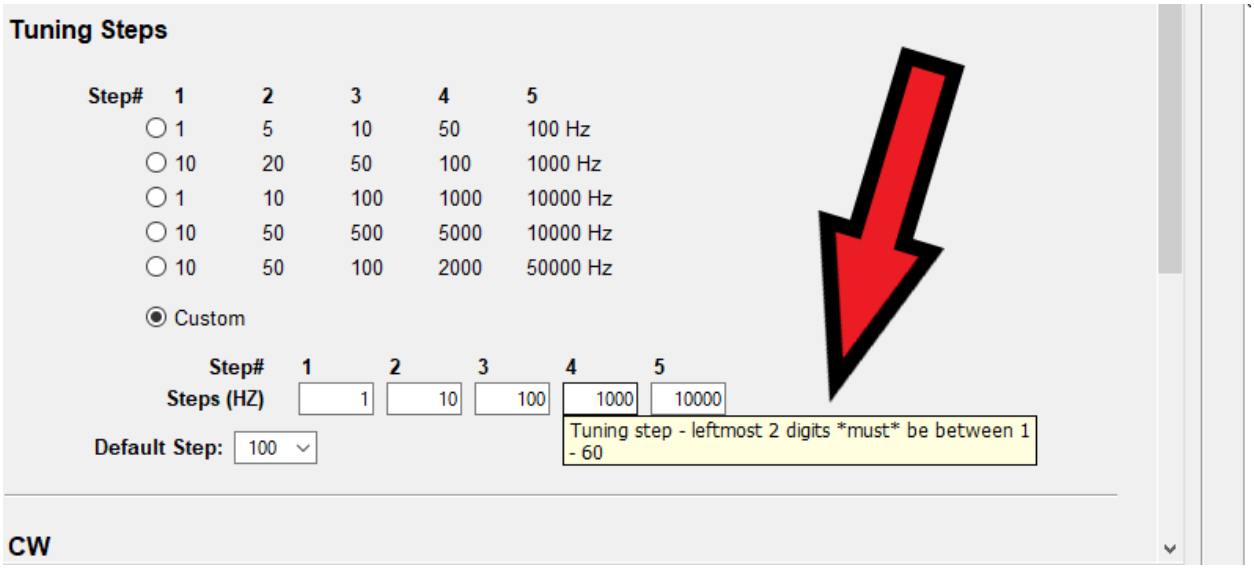
But don’t miss that button on the top right: “Copy Log To Clipboard”. If there is ever a problem and you want to help me find the problem, I will need this log. Click the button and paste it into an email.

Now for the fund part, you have selected your input source, clicked read, and now you are greeted with lots of information!



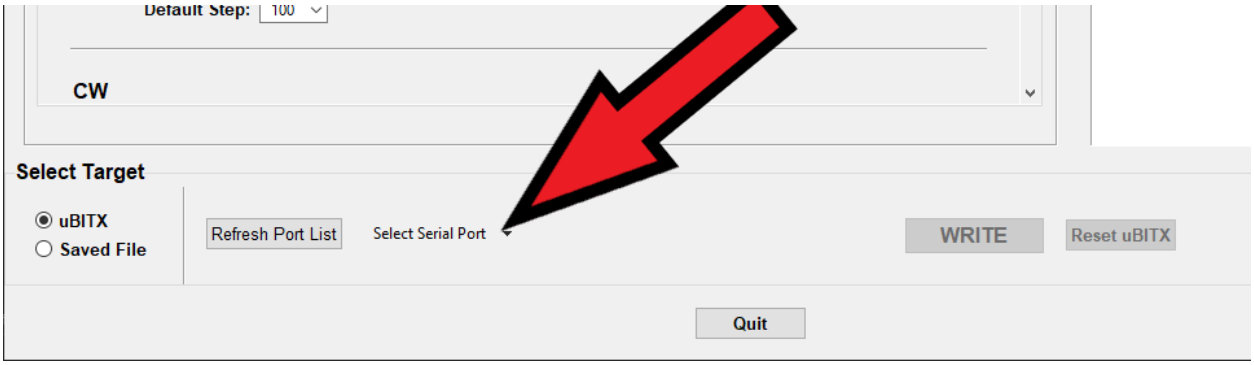

The big new is that that large center blank area is now filled with a  set of tabs that group similar settings together. In this case, the “General Settings” tab is active. Here you can set your callsign, adjust your default tuning step rate, and just off the screen, set your CW settings.

Future sections will focus on the contents of each tab separately. Just don’t forget the tooltops!



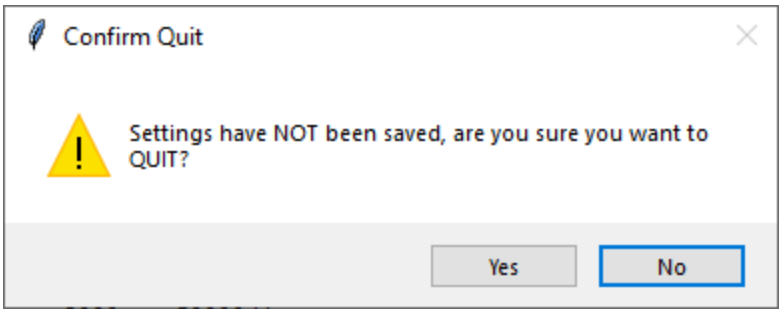

So eventually, you have tweaked every setting, and you want to save your work. Saving is just a reverse of the reading process. The box at the bottom of the software controls the writing.

In the screenshot, our friend the ugly red arrow points to the selection drop down for a list of com ports to which you will write the settings.




Generally, if you read your settings from a uBITX, you will just write them back to it. However, the separation of the input and output ports does allow you the possibility to read from one uBITX and write to another uBITX! **ONE CAUTION here, the positive protection against overwriting the calibration values when you read from one radio and write to another one is not in place. So make sure you record you calibration information before you update your second radio. That way you can restore them later from your notes.**

And while you are at it, you really should write your settings to a file so they are available to you in case of a future problem. Just select “Saved File”, click the arrow to bring up the file selection box, navigate where you want to save it. **THEN YOU MUST CLICK WRITE TO ACTUALLY SAVE THE SETTING TO THE FILE.** If you forget to click WRITE, you will be greeted with the following message when you try to exit:




The rest of this document will go thru each individual tab and talk about what settings are available and why you might want to use them.