uBTIX Settings Editor User Manual

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

Mark Hatch (AJ6CU)
3-6-2023

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.

Well, that was a good goal…

But as I was in the middle of porting the KD8CEC software to the Arduino BLE (I already had it working on the Arduino IOT), I discovered that the original uBITX Memory Manager refused to talk to the uBITX when I had an Arduino BLE installed. 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 Raspberry Pi Pico in an Arduino package) either. But then I found it liked the Teensy. So, what was going on here?

Unfortunately, Dr. Lee never released the source code to the uBITX Memory Manager. So I couldn't fix 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.

Consequently, I was left with no choice but to rewrite the uBITX Memory Manager.

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 save format of a human readable/editable XML file. However, 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 then just retrieiveing and storing bytes. I had to understand what every byte meant to present it to the user.

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 now available on macOS and Linux i(as well as Windows).

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

73

Mark (AJ6CU), March 6, 2023

Introduction

The uBITX is manufactured by HF Signals (https://www.hfsignals.COM/) using an Arduino Nano V3 MCU. This Nano is the heart of the uBITX. It has 32kbytes of flash memory and 1024 bytes of Electrically Erasable Programmable Read-Only Memory (EEPROM). The flash memory holds the firmware or software that runs the uBITX 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 was pretty basic and the plan all along was for other Hams to take the lead to extend it. 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 extended the original LCD offering and in later releases, provided a graphical user interface based on Nextion touchscreens. 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 see me equivalently referring to this as the "UX" or 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 settings 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, like all code does eventually, suffers from "virtual rot" and it was time for a rewrite.

This document is the user manual for the 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

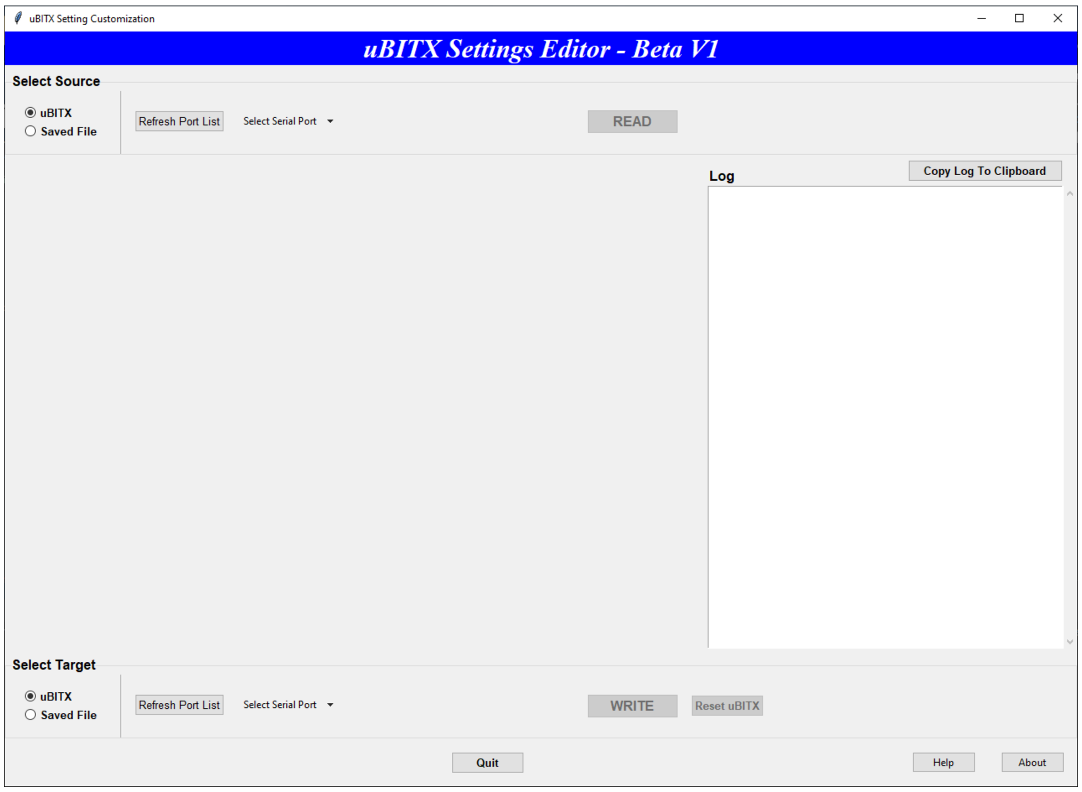
The next section will give you a basic orientation to the software and suggest a process on how you should use it. The software makes heavy usage of tooltips and provides larger hints adjacent to key functionality. I suspect that after this basic overview, you will be able to jump in and just use the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

Upon COMpleting the overview, we will take a slight detour and talk about how to install this software on Windows, macOS and Linux. In Beta 1, this software has not been "signed". This is a security process enforced on Windows and macOS that ensures the software is from the author ("signer") and has not been tampered with. As a result, the Beta 1 version will be slightly more involved to install on those platforms so we will take a step by step approach to make sure you are successful.

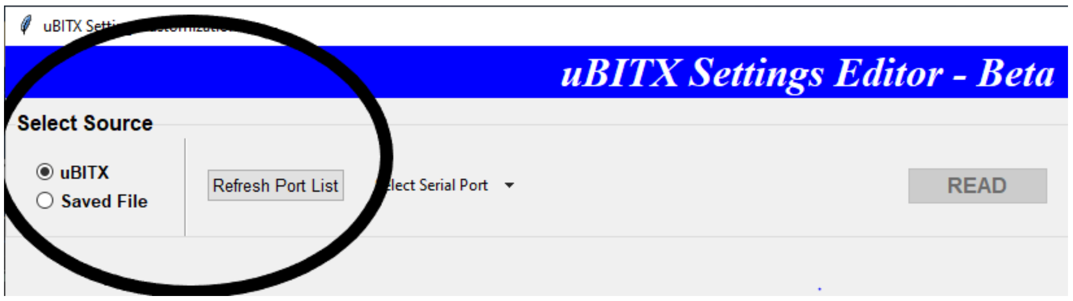
The final section, and the actual bulk of this manual, will focus on each basic area of functionality in detail. During this process, I attempt to share the knowledge that I gained during the reverse engineering process.

Quick Overview of the Settings Editor

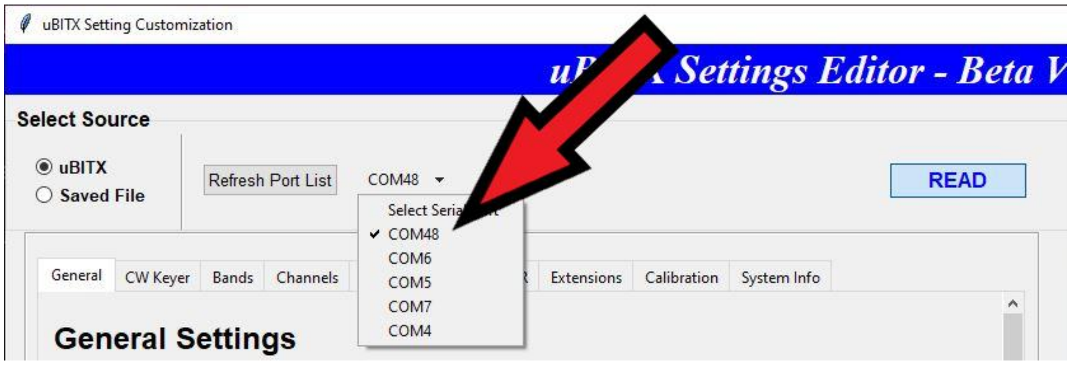
After you start the SE, you will be greeted by a screen that looks like:



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 is linked to your radio like in the following screenshot:

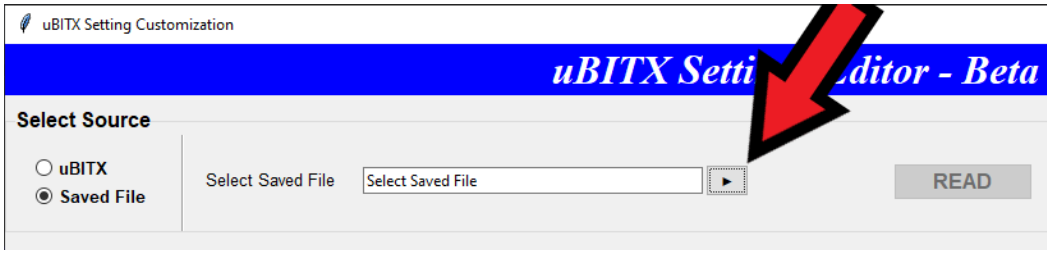


The (ugly) red arrow points out that the COM port listing box was clicked and that COM48 (in my case) was selected. Once the COM port is selected the READ button on the far right becomes active and you can read the settings into the *SE*.

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.

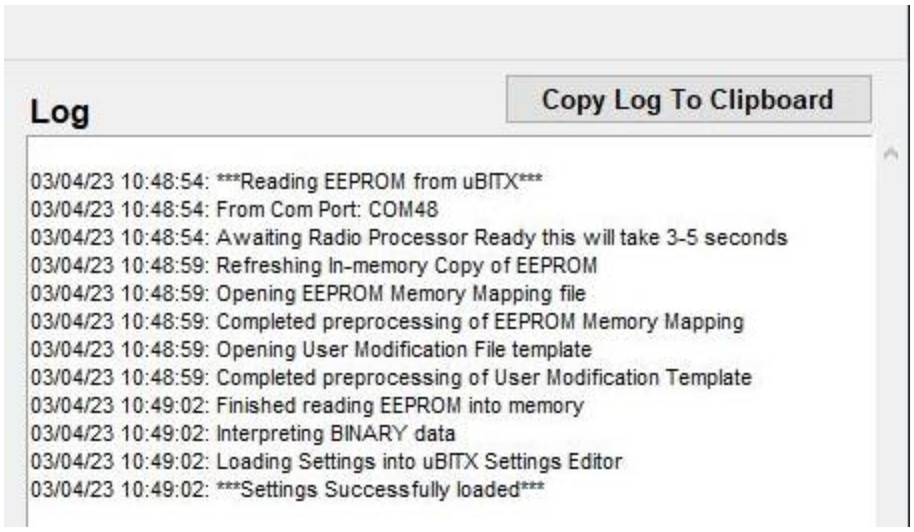
So why do you want to read your input from a file? Well, you really should backup your uBITX settings right? And perhaps when you switch processors in the future you can use this backup to restore your settings.

The screen below shows what happens when you select "Saved File". The COM port selection portion of the application is replaced by a file selector box. Clicking on that small arrow (pointed to by the red arrow) 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 the "Log" that is on the right side of the *SE*. For example, the following Log snipit was the result of reading the settings from a uBITX:

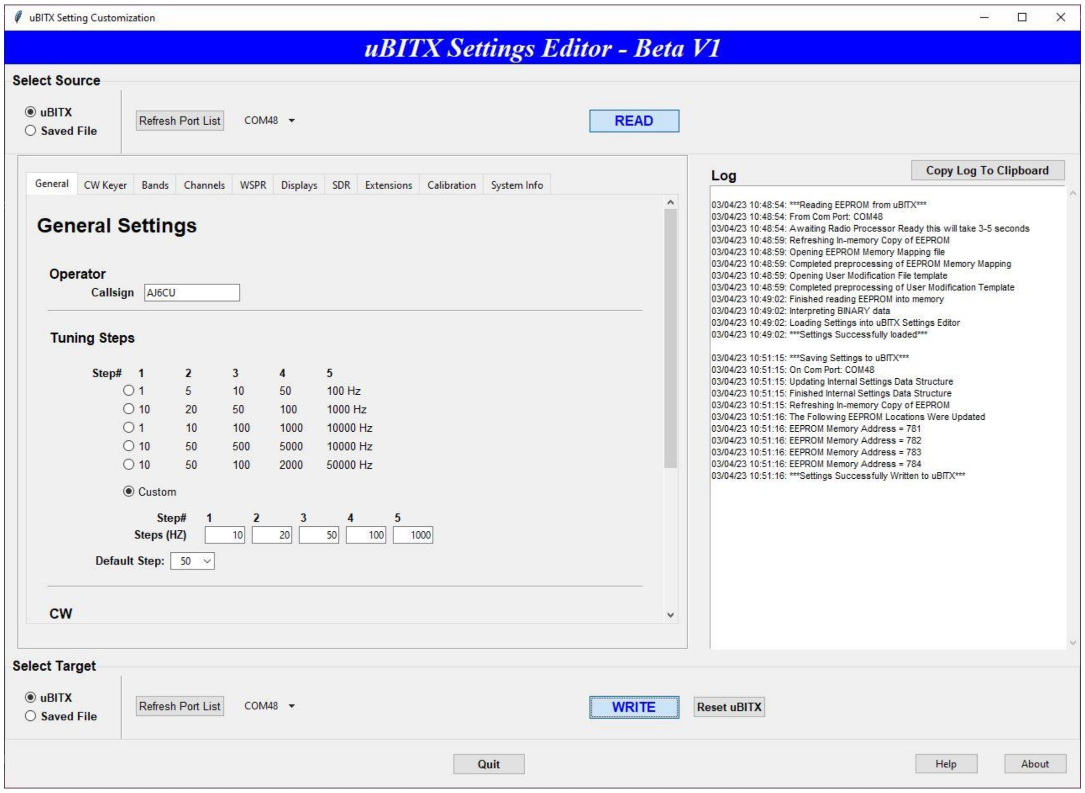


```
03/04/23 10:48:54: ***Reading EEPROM from uBITX***
03/04/23 10:48:54: From Com Port: COM48
03/04/23 10:48:54: Awaiting Radio Processor Ready this will take 3-5 seconds
03/04/23 10:48:59: Refreshing In-memory Copy of EEPROM
03/04/23 10:48:59: Opening EEPROM Memory Mapping file
03/04/23 10:48:59: Completed preprocessing of EEPROM Memory Mapping
03/04/23 10:48:59: Opening User Modification File template
03/04/23 10:48:59: Completed preprocessing of User Modification Template
03/04/23 10:49:02: Finished reading EEPROM into memory
03/04/23 10:49:02: Interpreting BINARY data
03/04/23 10:49:02: Loading Settings into uBITX Settings Editor
03/04/23 10:49:02: ***Settings Successfully loaded***
```

In this case, the second line of the log identifies the COM port where radio was connected (perhaps useful for the future). If you tried to enter data for a setting that was outside the legal bounds, a warning will appear here in red.

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 lots of information!



Select Source: [✓ uBITX] Saved File — COM48

General Settings

Operator
Callsign: AJ6CU

Tuning Steps

Step#	1	2	3	4	5
○	1	5	10	50	100 Hz
○	10	20	50	100	1000 Hz
○	1	10	100	1000	10000 Hz
○	10	50	500	5000	10000 Hz
○	10	50	100	2000	50000 Hz

[✓ Custom]

Step#	1	2	3	4	5
Steps (HZ)	10	20	50	100	1000

Default Step: 50

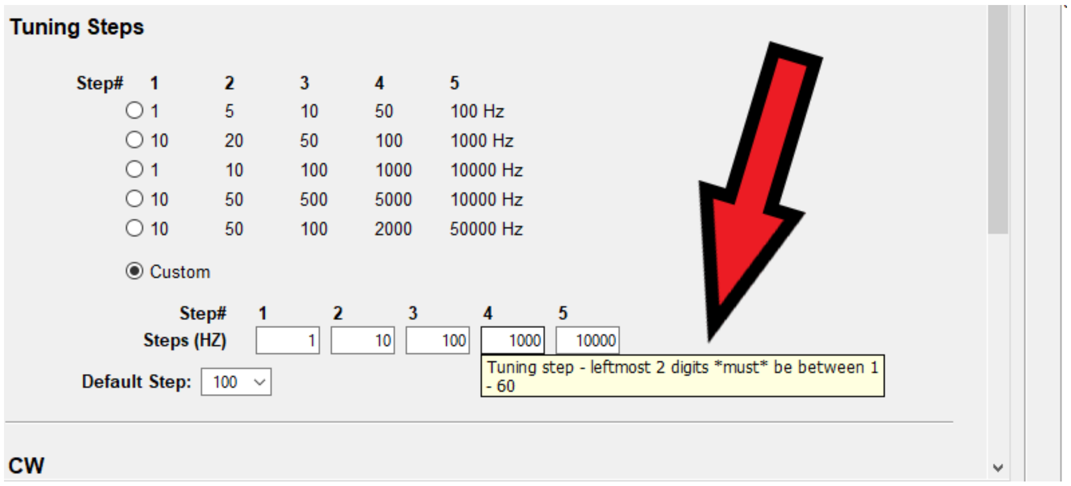
CW

Select Target: [✓ uBITX] Saved File — COM48

```
03/04/23 10:48:54: ***Reading EEPROM from uBITX***
03/04/23 10:48:54: From Com Port: COM48
03/04/23 10:48:54: Awaiting Radio Processor Ready this will take 3-5 seconds
03/04/23 10:48:59: Refreshing In-memory Copy of EEPROM
03/04/23 10:48:59: Opening EEPROM Memory Mapping file
03/04/23 10:48:59: Completed preprocessing of EEPROM Memory Mapping
03/04/23 10:48:59: Opening User Modification File template
03/04/23 10:48:59: Completed preprocessing of User Modification Template
03/04/23 10:49:02: Finished reading EEPROM into memory
03/04/23 10:49:02: Interpreting BINARY data
03/04/23 10:49:02: Loading Settings into uBITX Settings Editor
03/04/23 10:49:02: ***Settings Successfully loaded***

03/04/23 10:51:15: ***Saving Settings to uBITX***
03/04/23 10:51:15: On Com Port: COM48
03/04/23 10:51:15: Updating Internal Settings Data Structure
03/04/23 10:51:15: Finished Internal Settings Data Structure
03/04/23 10:51:15: Refreshing In-memory Copy of EEPROM
03/04/23 10:51:16: The Following EEPROM Locations Were Updated
03/04/23 10:51:16: EEPROM Memory Address = 781
03/04/23 10:51:16: EEPROM Memory Address = 782
03/04/23 10:51:16: EEPROM Memory Address = 783
03/04/23 10:51:16: EEPROM Memory Address = 784
03/04/23 10:51:16: ***Settings Successfully Written to uBITX***
```

The 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. (Remember the tooltips!)



	Step#	1	2	3	4	5
○		1	5	10	50	100 Hz
○		10	20	50	100	1000 Hz
○		1	10	100	1000	10000 Hz
○		10	50	500	5000	10000 Hz
○		10	50	100	2000	50000 Hz

● Custom

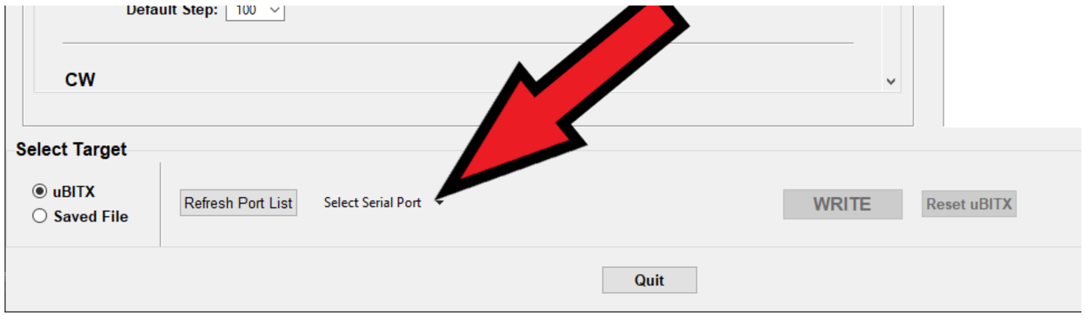
Step#	1	2	3	4	5
Steps (HZ)	1	10	100	1000	10000

Default Step: 100

Tuning step - leftmost 2 digits *must* be between 1 - 60

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.

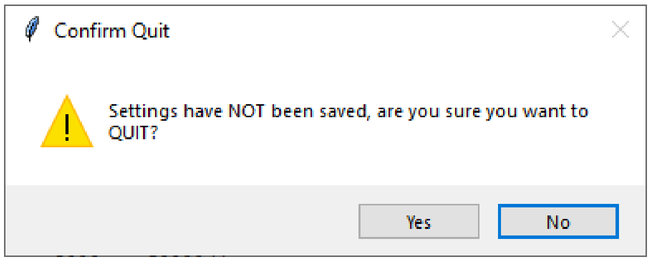
In the screenshot, the red arrow points to the selection drop down for a list of COM ports Select the one (generally the same as the input) and click WRITE.



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!

CAUTION: 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. **Remember to 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:



Settings have NOT been saved, are you sure you want to QUIT?

Hopefully, this overview has convinced you that this software isn't hard to use and that it might even be useful. Before taking a deep dive into the customizing the settings of each tab, lets get it installed on your machine!

Installation

This software was written using Python 3 and the Pygubu Designer for the Tkinter GUI toolkit. Without these two powerful tools, it would have taken much longer to complete. Furthermore, these tools made it trivial to support new platforms and operating environment. At this point, the following installation packages available:

- Windows 10 (x86 64 bit)
- macOS (Intel) (x86 64 bit)
- Linux (RaspberryPi ARM 32 and 64 bit)
- Source distribution for DYI builds

NOTE: BETA 1 IS NOT "SIGNED". THIS MEANS THAT THERE WILL BE SOME EXTRA STEPS TO CONVINCE MACOS AND WINDOWS THAT IT IS OK TO RUN THESE PROGRAMS. THE INSTRUCTIONS BELOW WILL HELP YOU NAVIGATE THIS PROCESS. HOWEVER, DEPENDING ON THE BROWSER, ANTIVIRUS PACKAGES, AND SPECIFIC OS AND VERSION YOU ARE RUNNING, THERE MAY BE SOME VARIATIONS FROM THESE STEPS. BETA 2 WILL INCLUDE SIGNED VERSIONS FOR WINDOWS AND MACOS.

The download page for these software packages is at:

https://github.COM/AJ6CU/uBITX-EEPROM-Manager/releases/tag/V2-beta-1

Scroll to the bottom of this page looking for the section labeled: "Assets"

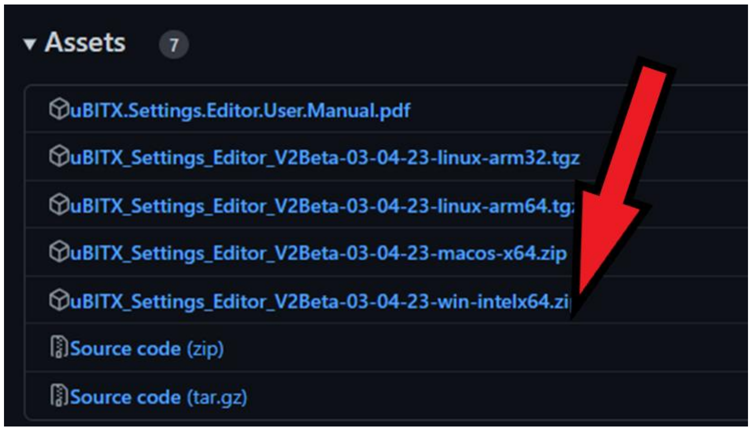
The asset itemized as "uBITX_Settings_Editor_User_Manul.pdf" is this document. The names of the other files should be sufficiently descriptive to let you select the correct package.

Windows 10 (x86 64 bit)

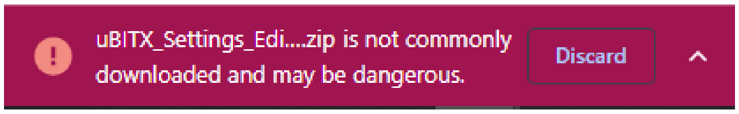
The Windows package is a Zip file. Once you open and extract it, you will find a standalone, one file executable that you can store it anywhere and just double click it to start.

However, you will first need to take the following steps to bypass Windows anti-malware protection. (Note: This was done on Windows 10, Windows 11 may vary in some ways.)

1. Download the file using Chome
   Click on the link pointed to by the Red Arrow.

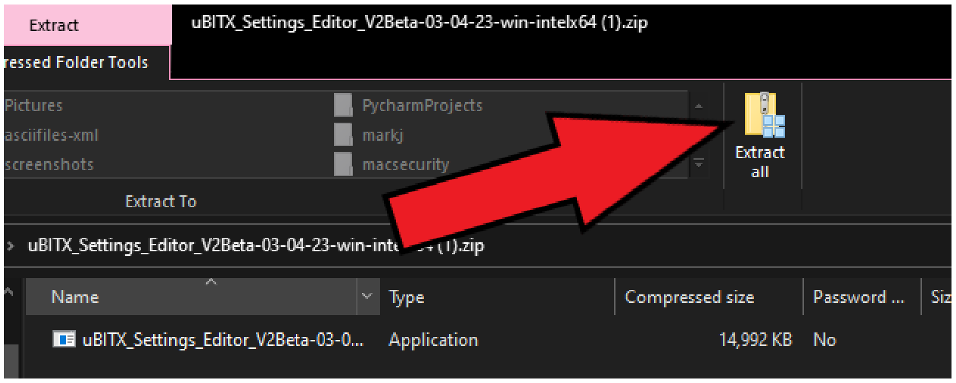

Assets 7
uBITX.Settings.Editor.User.Manual.pdf
uBITX_Settings_Editor_V2Beta-03-04-23-linux-arm32.tgz
uBITX_Settings_Editor_V2Beta-03-04-23-linux-arm64.tg
uBITX_Settings_Editor_V2Beta-03-04-23-macos-x64.zip
uBITX_Settings_Editor_V2Beta-03-04-23-win-intelx64.zi
Source code (zip)
Source code (tar.gz)

Chrome tries to protect you by providing you the following message.


uBITX_Settings_Edi....zip is not commonly downloaded and may be dangerous.

Click on the little up arrow to the right of the "Discard" button and select the "Keep" option.

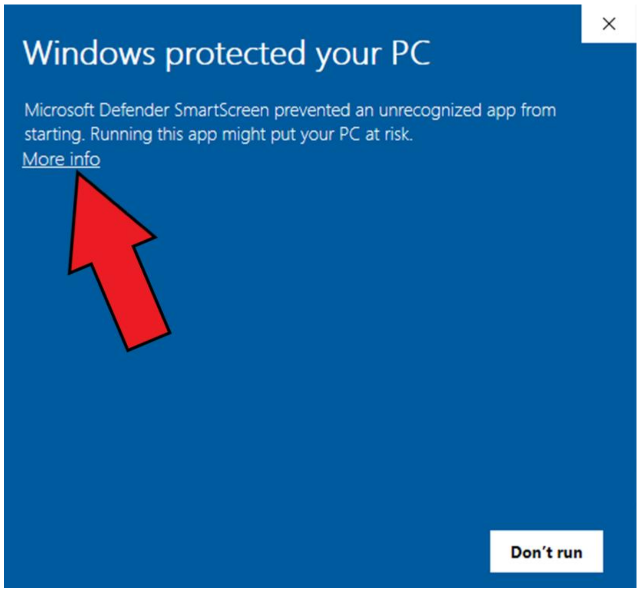
2. Expand the Zip
   Double click on the Zip file that you downloaded. Then click the "Extract all" button. I would recommend placing the extracted folder in its permanent location (I have a folder called "bin" in my home directory where I put stuff like this.)



uBITX_Settings_Editor_V2Beta-03-04-23-win-intelx64 (1).zip

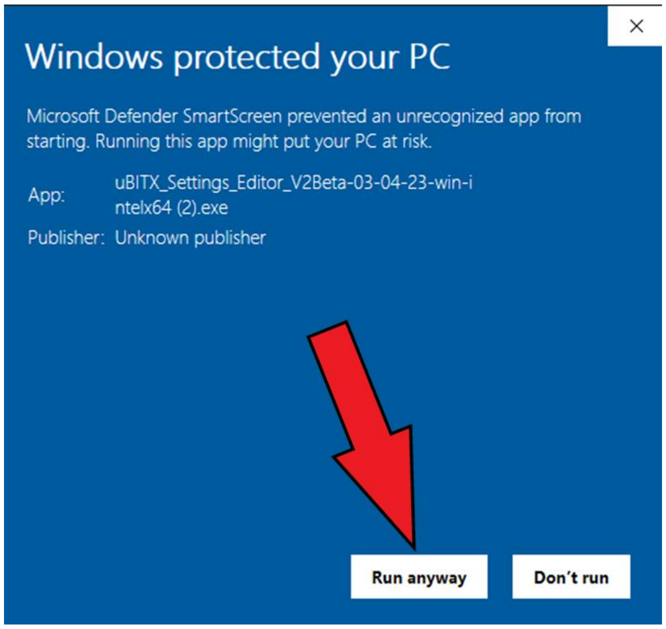
Name	Type	Compressed size	Password ...	Siz
uBITX_Settings_Editor_V2Beta-03-0...	Application	14,992 KB	No	

3. This will create a folder "uBITX_Settings_Editor_V2Beta-03-04-23-win-intelx64". Open it up and click uBITX_Settings_Editor_V2Beta-03-04-23-win-intelx64.exe. At this point, Windows should generate the following warning. Click on the "More info" link.


Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.
More info

4. This dialog box will now appear:



Windows protected your PC

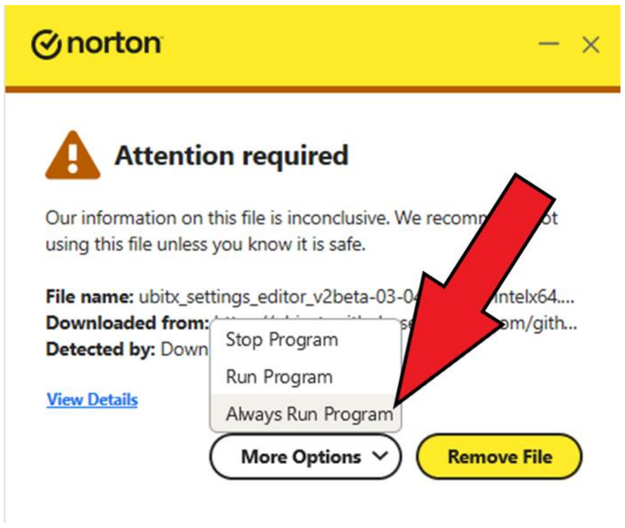
Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

App: uBITX_Settings_Editor_V2Beta-03-04-23-win-intelx64 (2).exe

Publisher: Unknown publisher

Click the "Run anyway" button. We are almost there!

5. I run Norton as my virus software and it tries to protect me with the following popup:



Attention required

Our information on this file is inconclusive. We recomm… …ot using this file unless you know it is safe.

File name: ubitx_settings_editor_v2beta-03-0… …intelx64....
Downloaded from: … …om/gith...
Detected by: Down…

Just click the "Always Run Program" and you are done!

6. I would recommend creating a shortcut to application and putting it in a convenient (perhaps your desktop). Just right click on the application, select "Create shortcut". Now drag the shortcut where you want it.

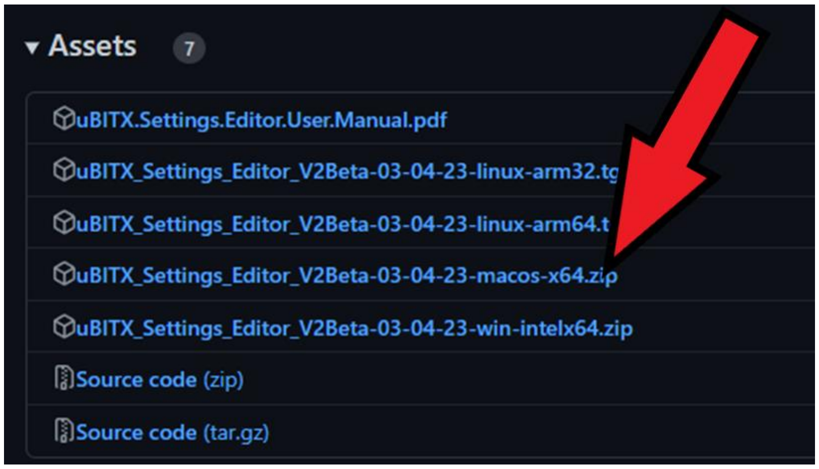
macOS

During this installation process, I was running macOS Monterey 12.4. If you are running a different release, your experience may vary. But the following steps worked for me:

1. Navigate to the release using either Chrome or Safari

   https://github.COM/AJ6CU/uBITX-EEPROM-Manager/releases/tag/V2-beta-1

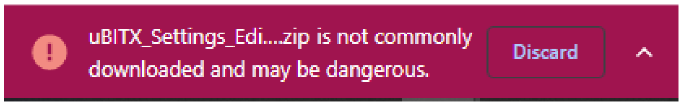
   Look for "Assets" at the bottom. Click on the link pointed to by the Red Arrow.

   Assets 7

   uBITX.Settings.Editor.User.Manual.pdf
   uBITX_Settings_Editor_V2Beta-03-04-23-linux-arm32.tg
   uBITX_Settings_Editor_V2Beta-03-04-23-linux-arm64.t
   uBITX_Settings_Editor_V2Beta-03-04-23-macos-x64.zip
   uBITX_Settings_Editor_V2Beta-03-04-23-win-intelx64.zip
   Source code (zip)
   Source code (tar.gz)

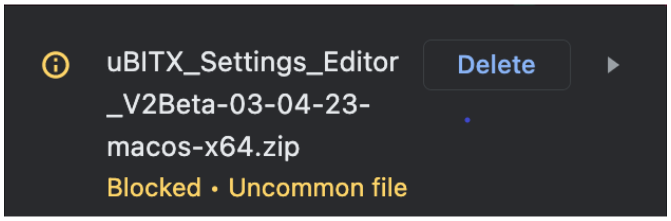
2. If you are using Chrome, then you will get the following message:

   uBITX_Settings_Edi....zip is not commonly downloaded and may be dangerous.

   See that little up arrow to the right of the "Discard" button? Click it and select the "Keep" option.

   If you are using Safari, it might first ask you "Do you want to allow downloads from Github". If it does, just click "yes". Next you might get the following message which is similar to the one from Chrome:

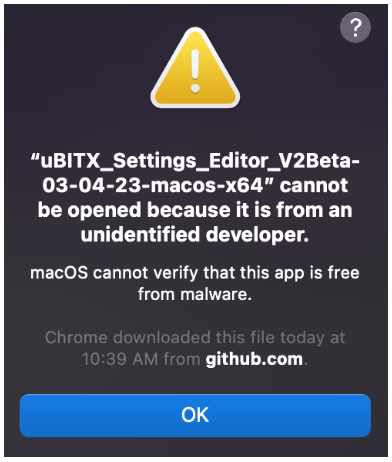
   

   uBITX_Settings_Editor_V2Beta-03-04-23-macos-x64.zip
   Blocked • Uncommon file

   Look for the right facing arrow to the right of Delete, click it and keep the file.

3. Safari will automatically unzip the file in your download area. If you used Chrome, you will need to navigate to your download folder and expand the zip file. At this point I would recommend you put the executable in its “permanent” home.

4. Now double click the executable. You should get the following error message because the application has not yet been signed. Click OK (you have no other choice.)

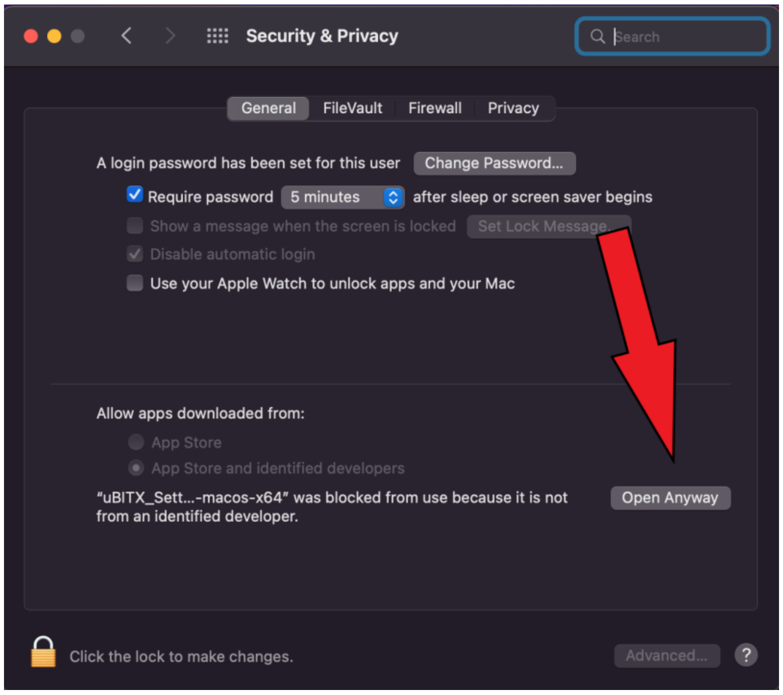


“uBITX_Settings_Editor_V2Beta-03-04-23-macos-x64” cannot be opened because it is from an unidentified developer.

macOS cannot verify that this app is free from malware.

Chrome downloaded this file today at 10:39 AM from **github.com**.

5. Next you will need to open the “Security and Privacy” control panel. Click on the “Apple” on upper right and then select “System Preferences”. Click on the Security & Privacy control panel. With the “General” tab, you should see the following:



A login password has been set for this user

[✓] Require password 5 minutes after sleep or screen saver begins

Show a message when the screen is locked

[✓] Disable automatic login

Use your Apple Watch to unlock apps and your Mac

Allow apps downloaded from:

App Store

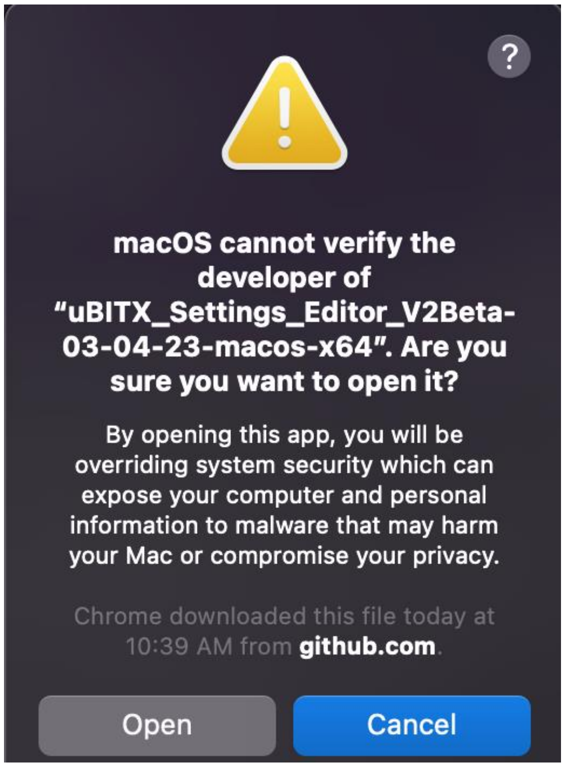
[● App Store and identified developers]

“uBITX_Sett...-macos-x64” was blocked from use because it is not from an identified developer.

Click the lock to make changes.

Click the “Open Anyway” button pointed to by the Red Arrow.

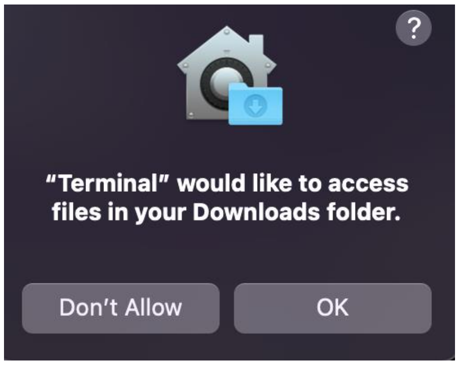
6. You should now get the following Dialog popup. Click “Open”


macOS cannot verify the developer of “uBITX_Settings_Editor_V2Beta-03-04-23-macos-x64”. Are you sure you want to open it?

By opening this app, you will be overriding system security which can expose your computer and personal information to malware that may harm your Mac or compromise your privacy.

Chrome downloaded this file today at 10:39 AM from **github.com**.

7. At this point you *may* see the following dialog popup:


“Terminal” would like to access files in your Downloads folder.

   Just click OK.
8. At this point, you should be greeted by the opening screen for the *SE*! The good news is that you have to do this once! From now on, it should just work.
9. As a footnote, you should see a console (terminal window) starting just before the *SE* Editor starts. Once you are done, just close this window. This is annoying and I will fix prior to Beta 2.

Linux (RaspberryPi ARM 32 and 64 bit)

If you are using the Linux distribution, you will be happy to hear that things are much easier for you.

1. Navigate to the release using browser.

   https://github.COM/AJ6CU/uBITX-EEPROM-Manager/releases/tag/V2-beta-1

   Look for "Assets" at the bottom. Click the distribution that matches your OS (either 32 bit or 64 bit) to download it.

2. Save the file (probably end up in your Downloads directory.

3. Open a folder to that location. Right click on the ".tgz" file you just downloaded, and then select either "Extract To" or "Extract Here".

4. In your target directory for your extracted folder, you should see a folder " uBITX_Settings_EditorV2Beta….". Open it.

5. Within the folder, you should see a file "uBITX_Setup_Editor". Double click it to start. When it asks you "Do you want to execute it? Select the "Execute" button.

6. It is a little annoying that this "what do you want to do" dialog keeps popping up. I suspect there is a solution…

Source Distribution for DYI Builds

One of the great things about using Python, is that it is very easy to take the source code to any machine and get the application working within minutes. I suspect that for many Linux users, the two distributions provided will **not work** for you. I also suspect that many Mac and Windows users would just like to build it themselves and avoid the hassle of convincing macOS or Windows that it is OK to execute the files provided.

If you are a source builder, take the following steps.

1. Make sure you have Python3 installed. Open a cmd or terminal window and try typing "python" at the prompt. When it starts, it will provide a Version#. If it says Python 2, type exit() and when you get back to the terminal prompt, try typing "python3". If you get a "not found" error message at this point, download and install the latest python from www.python .org.

2. Make sure the following extensions to Python are installed by executing the following commands (this is done at the shell prompt and **not within** python interpreter:

```
pip3 install lxml
pip3 install pyserial
pip3 install bitarray
pip3 install pygubu
```

3. Download the source distribution (see the project assets) or clone the repository

4. Navigate into the uBITX-EEPROM-Manager /uBITX_Seetings_Editor and type:
   python3 uBITX_Setting_Editor.py
5. You should see the *SE* start!

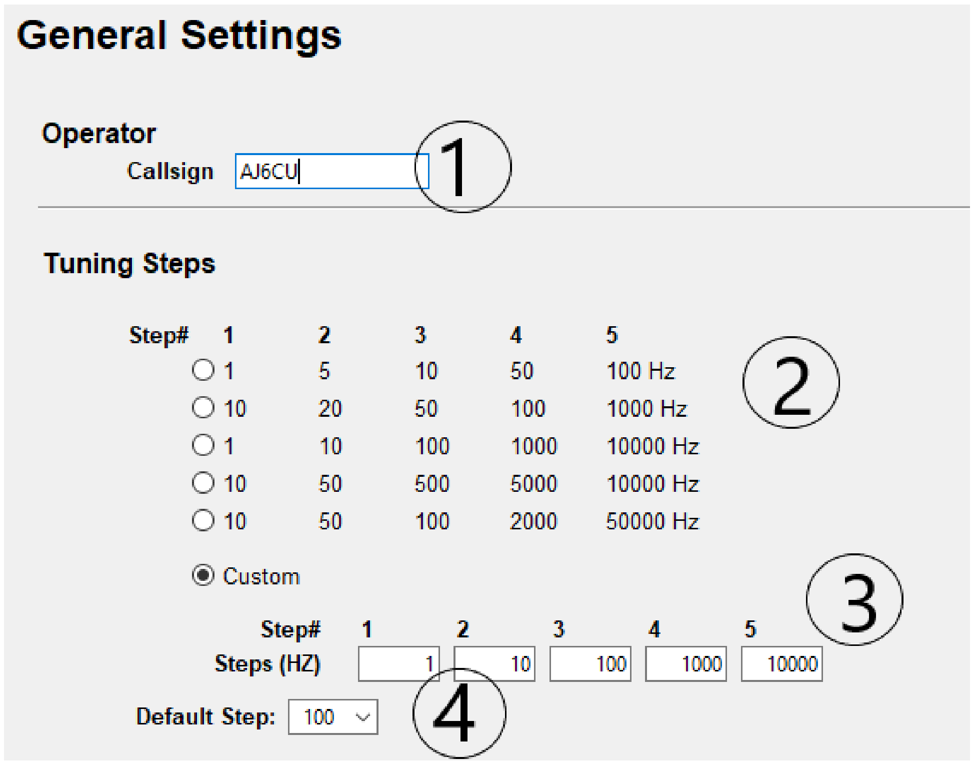
Indepth Review of the Functionality of the uBITX Settings Editor

The primary workspace for modifying the settings of your uBITX is a set of tabs -- much like dividers in a 3-ring binder -- that is in the middle of the application. Currently, there are tabs for:

- General – settings that most users will need to modify
- CW Key – controls the automated CW keyer used contesting
- Bands – defines the available frequency bands
- Channels – allows the creation and management of frequently used frequencies
- WSPR – builds messages and select frequencies for a WSPR beacon
- Displays – manage LCD display and customize their presentation of information
- SDR – configures the uBITX to use an SDR dongles
- Extensions – supports the edition of function keys, customization of the LPF and
- Calibration – key hardware values that govern frequency accuracy, CW key interpretation and S-meters
- System Info – documents installed firmware version, factory values and last used frequencies and modes

General Tab

The General Tab collects the first settings that a user *should* want to change. Ideally, a user should tweak these settings, save them to the uBITX (and hopefully a backup file).



General Settings

Operator

Callsign: AJ6CU

Tuning Steps

Step#	1	2	3	4	5
○	1	5	10	50	100 Hz
○	10	20	50	100	1000 Hz
○	1	10	100	1000	10000 Hz
○	10	50	500	5000	10000 Hz
○	10	50	100	2000	50000 Hz

[✓ Custom]

Step#	1	2	3	4	5
Steps (HZ)	1	10	100	1000	10000

Default Step: 100

By the Numbers

The screenshot above is the *top half* of the General Settings tab. The numbers 1,2,3,4 are used to highlight key features. We will use a similar approach with all the other tabs and screenshots.

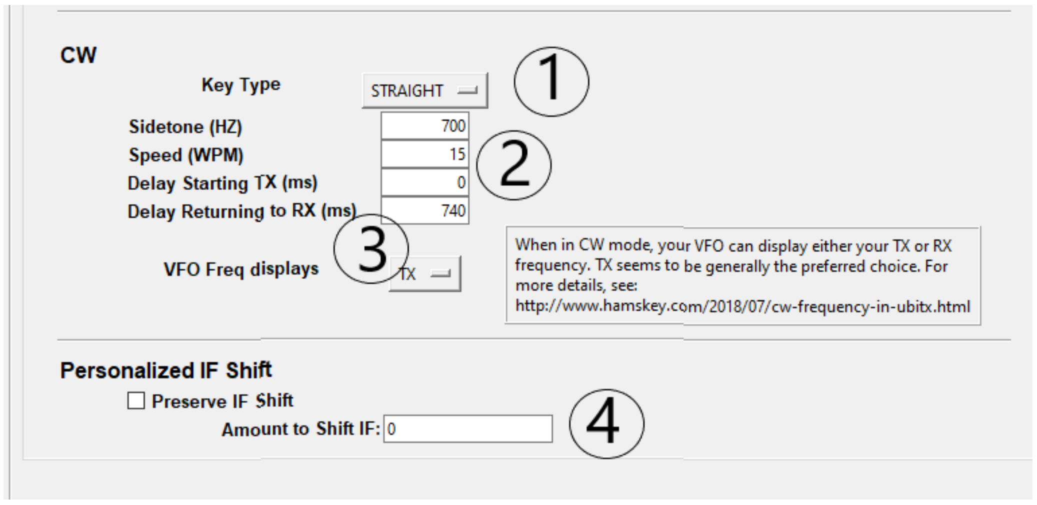
1. The first thing any Ham should do is to put his Callsign into his radio. You do this on the uBITX by entering your data in this field. As the tooltip indicates, you can enter up to 18 characters. This should leave enough room for callsigns that need to be qualifies (i.e. AJ6CU/MM if this radio was being used in a portable mode)

2. Most Hams spend a lot of time searching the dial. The "2" is next to the "standard" tuning rates. Click the circle next to the tuning rate that seems to best me your needs. You can switch between the rates either using the UX of the Nextion based systems or by pushing in the encoder knob for a "long press" and then turning it to the desired step when prompted.

3. KD8CEC does allow you to adjust the tuning rate of your radio. But because of the (ugly) way that the tunings steps are saved in the EEPROM, there are limits to the types of rates you can input. The rule is that the first two digits (left most) for every tuning step, must be between 0 and 60. So for example, in the above screenshot, 50 would be a fine step for tuning step 3. And even 60. But 61 would be forbidden. Similarly, in tuning step#4, 600 would be fine, but 601 would not. This strange constraint was the result of a devious byte saving trick where one byte holds both the significant digits (0-60) and the multiplier (i.e. x1, x10, x100, x1000, etc.) Do yourself a favor and just select one of the standard step ranges. They will be fine.

4. The final field of interest is a drop down box that allows you to specify the default tuning rate when the radio is turned on. In this case, turning the knob one encoder click will increase or decrease the frequency by 100hz.

The screenshot below is of the bottom half of the General Tab (if you cannot see it, use the scrollbar on the right to move it into view).



When in CW mode, your VFO can display either your TX or RX frequency. TX seems to be generally the preferred choice. For more details, see: http://www.hamskey.com/2018/07/cw-frequency-in-ubitx.html

By the Numbers

1. The first choice you need to make is whether you will be using a Straight Key or a paddle. And if a paddle, are you using Iambica A or Iambica B keying modes.

2. The first two options in this area are pretty clear: what should be the frequency of the CW sidetone and what speed do you want your keyer to be set at. The next two settings are delays. The first is the delay *before* starting transmission. Generally you want this to be zero. The other setting, Delay Returning to RX is probably more commonly used. If you are a little hesitant with your keying, you could find your uBITX going in and out of TX mode between words or characters. This allows you to add some delay to avoid the annoying bang of the TX relays.

3. This is an interesting and sometimes controversial setting. When you send CW, your actual TX signal is offset from the frequency of the radio by the amount of your sidetone (in this case 700). This is pretty much unique to CW. The question that gets debated is whether the frequency shown on the VFO is the TX frequency or the RX frequency. The general consensus of contesters and DX folks seems to be that they want to know where they are transmitting, This drop down box allows you to control whether your VFO shows the TX frequency (RX +/- Sidetone depending on USB/LSB) or the RX frequency. There is an articled called out in the note on this that you can read on Hamskey site. There is also a lot of discussion about this in the archives of BITX20 group and on ubitx.net

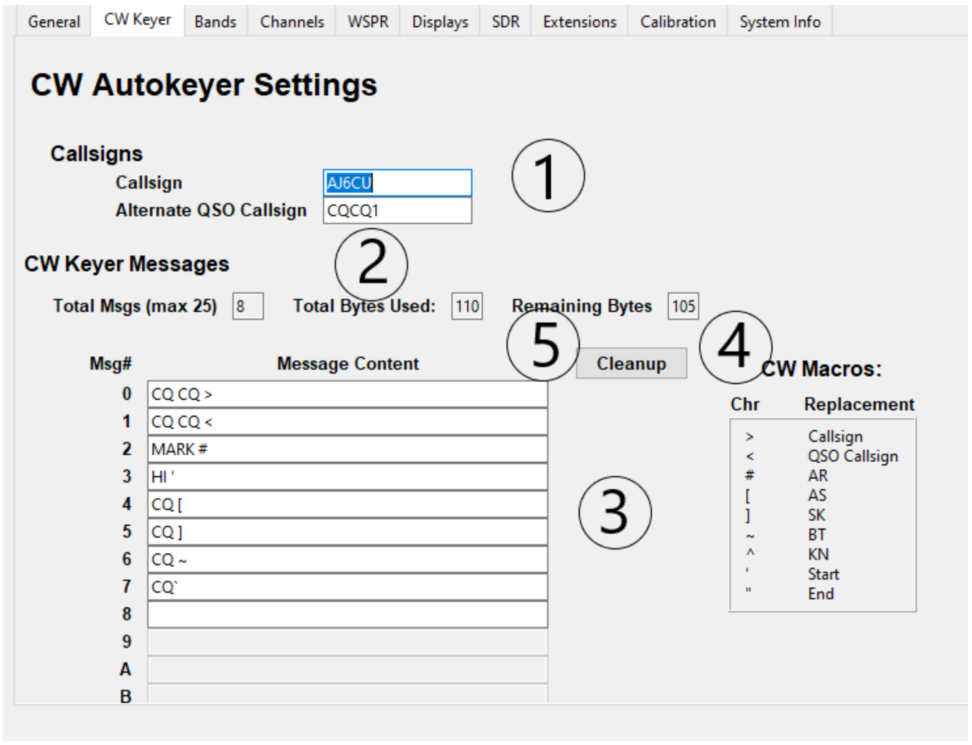
4. By adjusting the IF Shift, you can tune the radio to make things sound better, especially with SSB voice. If you find yourself playing with the IF Shift a lot, you might want to program the offset in and click the "Preserve IF Shift" box. This way when you turn on your radio, it is tailored to your preference.

CW Keyer

The KD8CEC software provides a fairly functional keyer that can be tailored for contesting. Although not as sophisticated as some (e.g., doesn't have any automated serial number counting), it does provide for character substitution and up to 25 messages! But from a practical viewpoint, 25 is probably far too many handle and the total memory available for keyer messages is on the order of 210 characters. So realistically, you will out of memory before you run out of available messages.

By the Numbers

1. The callsign you entered in the General Settings is repeated up here at the top. (It is read-only so you cannot change it here. Go back to the General Settings if you want to change it.) This tab does provide the option of an alternative callsign that you can use. Perhaps your contesting from a remote site and want to signify that with something like AJ6CU/5. Like your normal callsign it is limited to 18 characters.
2. Since EEPROM memory is limited, the line identified by "2" tracks 3 key aspects: 1. How many messages are active, 2. Total bytes used, and perhaps most importantly 3. Remaining bytes
3. These are the CW messages that have been saved. Everytime you enter the last one (for example we just entered the "CQ" in message #7, the next line (#8) becomes open. This was done this way to help you save memory since every active message has a 2 byte overhead.



CW Autokeyer Settings

Callsigns

Callsign	AJ6CU
Alternate QSO Callsign	CQCQ1

CW Keyer Messages

Total Msgs (max 25) 8 Total Bytes Used: 110 Remaining Bytes 105

Msg#	Message Content
0	CQ CQ >
1	CQ CQ <
2	MARK #
3	HI '
4	CQ [
5	CQ]
6	CQ ~
7	CQ`
8	
9	
A	
B	

CW Macros:

Chr	Replacement
>	Callsign
<	QSO Callsign
#	AR
[	AS
]	SK
~	BT
^	KN
'	Start
"	End

4. This is the list of macros you can use in your CW message. Even though a ">" expands to your full callsign, you only use one byte (for the character ">") in your message. Although these macros are convenient, they are especially useful for saving Keyer memory.

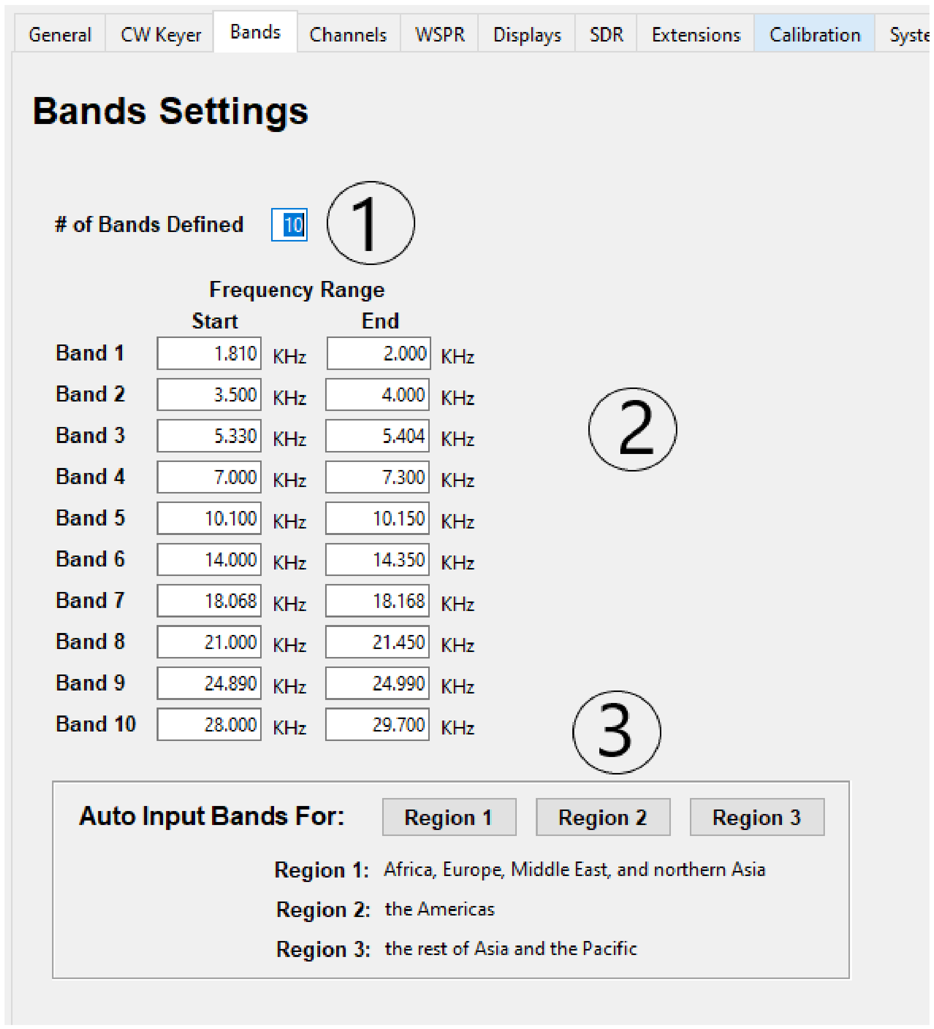
5. This button (CLEANUP) will eliminate empty messages and move them all up to the front. For example, suppose I decided to delete message 5 "CQ]". This would leave an empty slot in message #5. Hitting the CLEANUP button, will move 6 into slot 5, 7 into slot 6, delete the blank line at 8, and recovery the extra overhead bytes. You get a neater looking set of messages (especially important if you are keying via the rotary encoder), and you free up some bytes.

Bands

This tab is used to set the bands that are appropriate for your region. Since the uBITX is sold and assembled worldwide, making sure that you are in the legal portion of the band in your part of the world is important. These setting directly impact the bands you see as you go +band or -band. So let's do the numbers for the screen shot below showing the **top half** of the Bands tab.

By The Numbers

1. This is the number of active (defined) bands that your uBITX knows about. If you put 9 into this field, it will only know about the first 9 bands even though you might have a frequency in slot 10. And if the number is greater than the number of bands you have defined, you will have to cycle thru those empty bands to get back to your favorite.



Bands Settings

# of Bands Defined: 10

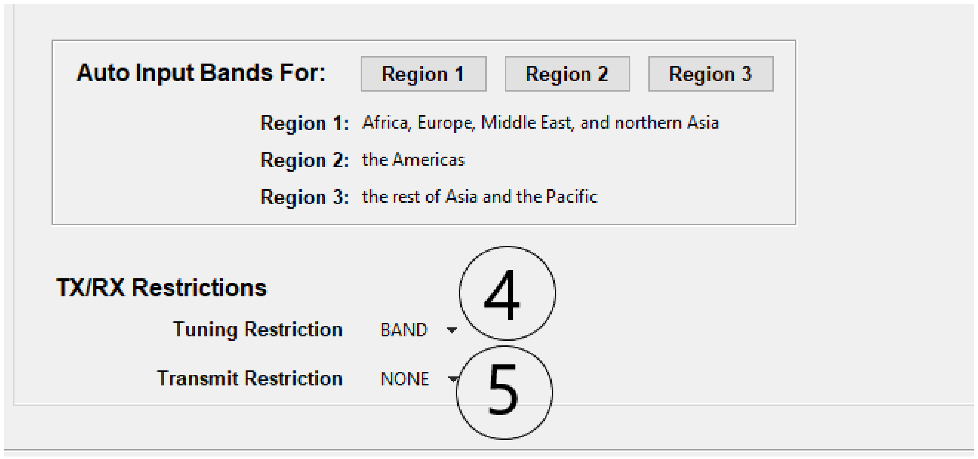
	Frequency Range Start		Frequency Range End	
Band 1	1.810	KHz	2.000	KHz
Band 2	3.500	KHz	4.000	KHz
Band 3	5.330	KHz	5.404	KHz
Band 4	7.000	KHz	7.300	KHz
Band 5	10.100	KHz	10.150	KHz
Band 6	14.000	KHz	14.350	KHz
Band 7	18.068	KHz	18.168	KHz
Band 8	21.000	KHz	21.450	KHz
Band 9	24.890	KHz	24.990	KHz
Band 10	28.000	KHz	29.700	KHz

Region 1: Africa, Europe, Middle East, and northern Asia
Region 2: the Americas
Region 3: the rest of Asia and the Pacific

2. For each Band, you enter the beginning and ending frequency **in KHz (not MHz).**

3. It is a real pain, and definitely error prone to find the definitive frequency map for your area and then enter 9 or 10 bands of numbers. So like in the original uBITX Memory Manager, this software will automatically fill this in at the push of the button.

4. On the screenshot below, the bottom part of the Band map tab provides a set of options to control where you can Tune to (Tuning Restriction) as well as where you can transmit. #4 deals with RX and there are two options: Band and None. Setting it to **BAND** means that when you using the encoder to change bands (short press, followed by another press on the “Band Select?” menu) you will move from band to band (i.e. 40m, 30m, 20m, etc.) If you select **NONE, when you go into “Band Select” you are actually moving the VFO up or down by 100k Hz and you can easily end up between legal HAM bands.**



Auto Input Bands For: Region 1 Region 2 Region 3

Region 1:	Africa, Europe, Middle East, and northern Asia
Region 2:	the Americas
Region 3:	the rest of Asia and the Pacific

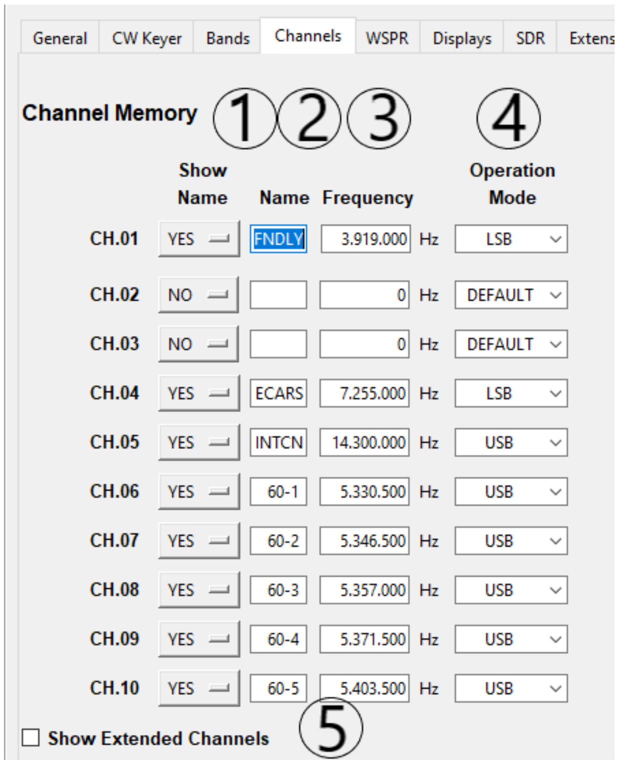
TX/RX Restrictions

Tuning Restriction	BAND
Transmit Restriction	NONE

5. You can also control where you can transmit. There are two options here: NONE and HAM. NONE means you can transmit anywhere while HAM means when you PTT or hit the key, you will only go into transmit if you are within a HAM band.

Channels

KD8CEC provides the useful concept of frequency channels. This allows you to identify and hop between up to 20 channels (10 with names, 10 just by numbers) and their respective mode (CWL,CWU, LSB, USB).



Channel Memory

	Show Name	Name	Frequency		Operation Mode
CH.01	YES	FNDLY	3.919.000	Hz	LSB
CH.02	NO		0	Hz	DEFAULT
CH.03	NO		0	Hz	DEFAULT
CH.04	YES	ECARS	7.255.000	Hz	LSB
CH.05	YES	INTCN	14.300.000	Hz	USB
CH.06	YES	60-1	5.330.500	Hz	USB
CH.07	YES	60-2	5.346.500	Hz	USB
CH.08	YES	60-3	5.357.000	Hz	USB
CH.09	YES	60-4	5.371.500	Hz	USB
CH.10	YES	60-5	5.403.500	Hz	USB

☐ Show Extended Channels

By the Numbers

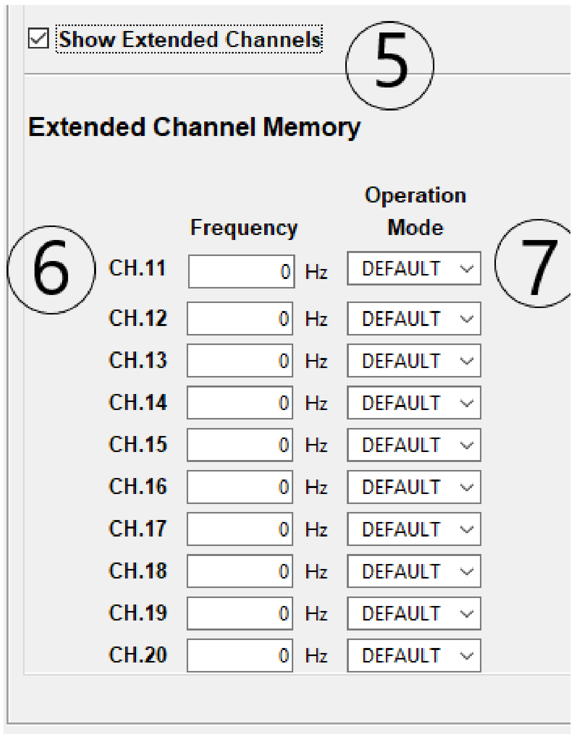
1. The first 10 channels can be assigned a 5-character name. But you might not need or want the name displayed on your screen (LCD or Nextion). This menu allows you to say **YES** show the name or **NO** just use the Channel# instead.

2. Enter a 5-character name here. Any characters are allowed.

3. This is where you would enter the frequency in HZ.

4. This is a drop-down menu that allows you to select LSB, USB, CWL, CWU, DEFAULT).

5. So where are the additional 10 channels? Click this box to see them and then use the scrollbar on the right to make them visible.



☑ Show Extended Channels

Extended Channel Memory

	Frequency		Operation Mode
CH.11	0	Hz	DEFAULT
CH.12	0	Hz	DEFAULT
CH.13	0	Hz	DEFAULT
CH.14	0	Hz	DEFAULT
CH.15	0	Hz	DEFAULT
CH.16	0	Hz	DEFAULT
CH.17	0	Hz	DEFAULT
CH.18	0	Hz	DEFAULT
CH.19	0	Hz	DEFAULT
CH.20	0	Hz	DEFAULT

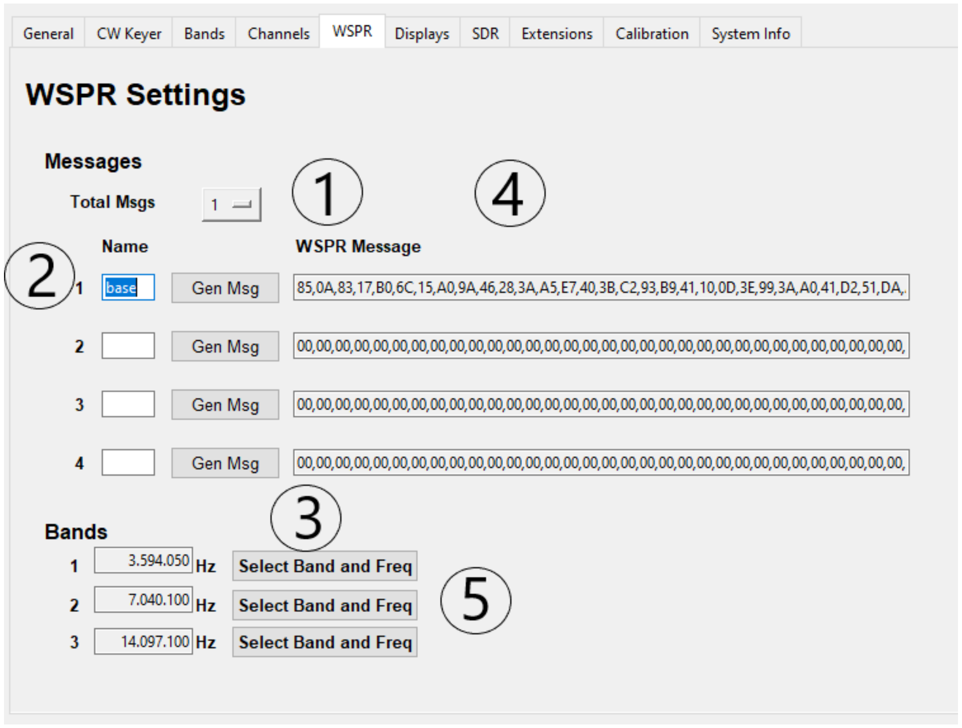
6. Here are the other 10 Channels! When you access them on your uBITX they are simply names CH11, CH12, CH13, etc. Enter the frequency of the Channel in Hz.

7. Then choose the operational mode. Same options as before: LSB,USB,CWL,CWU, DEFAULT.

WSPR

Going back thru the archives of the BITX20 group, there a lot of excitement about the addition of WSPR (Weak Signal Propagation Reporter) to the KD8CEC software. The neat thing about this feature was that it is standalone and did not require that it was being driven from an attached PC. Basically, you can define up to 4 messages, and three frequency and then transmit on any combination of them.

The *SE* kept the UX format of the WSPR subsystem originally implemented in the uBITX Memory Manager. the same as it was presented originally. So existing users will make the transition easily.



WSPR Settings

Messages

Total Msgs: 1

	Name		WSPR Message
1	base	Gen Msg	85,0A,83,17,B0,6C,15,A0,9A,46,28,3A,A5,E7,40,3B,C2,93,B9,41,10,0D,3E,99,3A,A0,41,D2,51,DA,.
2		Gen Msg	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,
3		Gen Msg	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,
4		Gen Msg	00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,

Bands

1	3.594.050 Hz	Select Band and Freq
2	7.040.100 Hz	Select Band and Freq
3	14.097.100 Hz	Select Band and Freq

By the Numbers

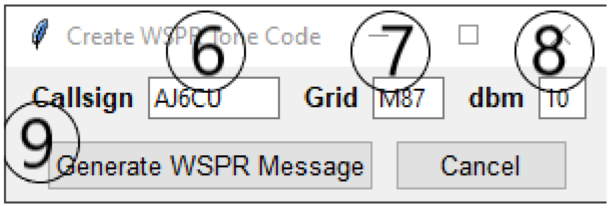
1. This drop down identifies how many messages will be made available on your uBITX. If you select 1 (as in this screenshot), you will only have one message available (even if you have more messages defined.)

2. You can name your WSPR message here using any 5 chracters. That makes it easier to select the preferred message when you are operating.

3. Click the “Gen Msg” msg button to bring up a dialog box that will collect the information needed and generate the actual WSPR message.

4. This is actual WSPR message in Hex format. The KD8CEC software that runs on the uBITX will take this message and convert it to the 2 bit frequency shift that is used to transmit the WSPR message.

5. The WSPR frequencies that are going to be used are selected here. You can choose up to 3 different frequencies. Then after you choose which message to send, you choose the frequency. Click “Select Band and Freq” to bring up a dialog box that helps you select the band and specific frequency.

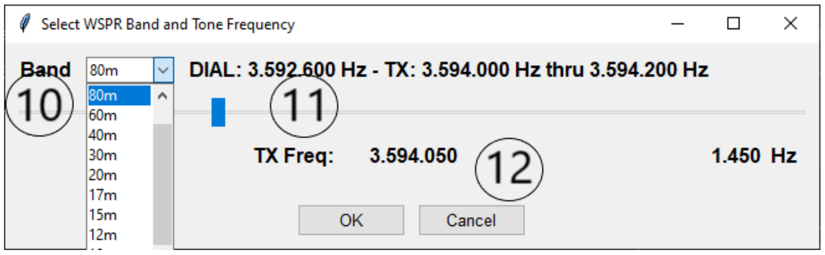


6. This dialog is used to create the WSPR message. The first thing to enter is your callsign. The WSPR protocol is very strict and there is a maximum of 6 characters allowed with numbers having to be in specific positions. Just enter your call sign here.

7. Enter the first 4 characters of your maidenhead grid square. For example, mine is CM87.

8. If you like, you can adjust the reported dbm to match your TX output.

9. Once all the information is entered, hit the “Generate WSPR Message” button to generate the message and close this dialog box.


Select WSPR Band and Tone Frequency
Band
80m
10
80m
60m
40m
30m
20m
17m
15m
12m
DIAL: 3.592.600 Hz - TX: 3.594.000 Hz thru 3.594.200 Hz
11
TX Freq:
3.594.050
12
1.450 Hz
OK
Cancel

10. This dialog pops up to help you select your TX frequency. The drop down on the left (#10) allows you to select the band you will transmit on. Do this first.

11. The slider allows you to adjust your transmit frequency to maximize its opportunity to be received by another station.

12. The actual TX frequency, Selected band + offset is reported here. Ideally, you should see your transmission reported very near this frequency. If not, you might want to consider tuning your Master calibration number.

Displays

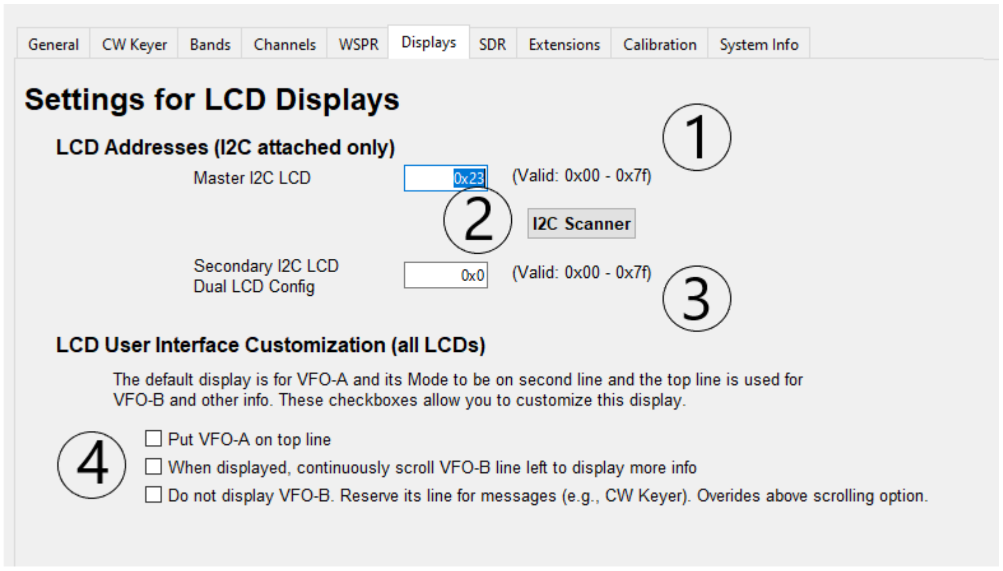
This tab only addresses LCD displays. If you have a Nextion display, just keep moving on to the next section.

There are two types of connections you might have between your LCD and the Raduino (where the processor sits on an uBITX). The original connection was “parallel” where 6 digital lines (4 data and 2 control) were connected to the LCD. This approach occupied a lot of pins and the Nano didn’t have extra pins. So early on, some folks went to the I2C based LCD’s. This immediately freed up the 6 digital pins for other things. If you never did anything with your display, you have a parallel connected LCD. If you bought your uBITX second hand and you are not sure, then you can figure this out by seeing where the LCD is connected. If it is plugged directly into the Raduino board, it is parallel.

The knowledge of whether you have a parallel or I2C LCD is important for selecting the right values on this tab. If you have an I2C, you might need to set the I2C address (or addresses if you have two I2C LCD’s). In which case, the upper part of the screenshot below is relevant.

By the Numbers

1. You can enter the address of your primary (“Master”) LCD here. Unless you pulled out your soldering iron and changed some jumpers, the standard address is 0x23 as shown in the screenshot. This is true for both 16x2 as well as 20x4 (20 characters by 4 lines) LCD. BTW, unless you are going for small and light, the 20x4 screen and the extra info you get is really nice!



Settings for LCD Displays

LCD Addresses (I2C attached only)

Master I2C LCD 0x23 (Valid: 0x00 - 0x7f)

Secondary I2C LCD
Dual LCD Config 0x0 (Valid: 0x00 - 0x7f)

LCD User Interface Customization (all LCDs)

The default display is for VFO-A and its Mode to be on second line and the top line is used for VFO-B and other info. These checkboxes allow you to customize this display.

☐ Put VFO-A on top line
☐ When displayed, continuously scroll VFO-B line left to display more info
☐ Do not display VFO-B. Reserve its line for messages (e.g., CW Keyer). Overides above scrolling option.

2. If you are not sure of the address, then you can always click the I2C Scanner button and take a peek at your I2C bus. More on this on the next screenshot analysis.

3. One of the fun things you can do if you have a second 16x2 LCD attached. (This would be as an alternative to a 20x4.) I don’t recommend this solution as it is actually physically larger than a single 20x4 and it is just hard to mount two 16x2 screens.

4. This are deals with customizing how the information on your LCD appears. The first option ("Put VFO-A on top line") moves VFO-A from its default bottom position to the top.

   Shouldn't be a surprise that a 16x2 LCD has limited amount of space to display information. You might check the scrolling option for VFO-B and see if this works for you. (I find the movement distracting and leave it unchecked.)

   The final checkbox allows you to eliminate the display of VFO-B completely and let KD8CEC repurpose this space for other messages.

I2C Scanner

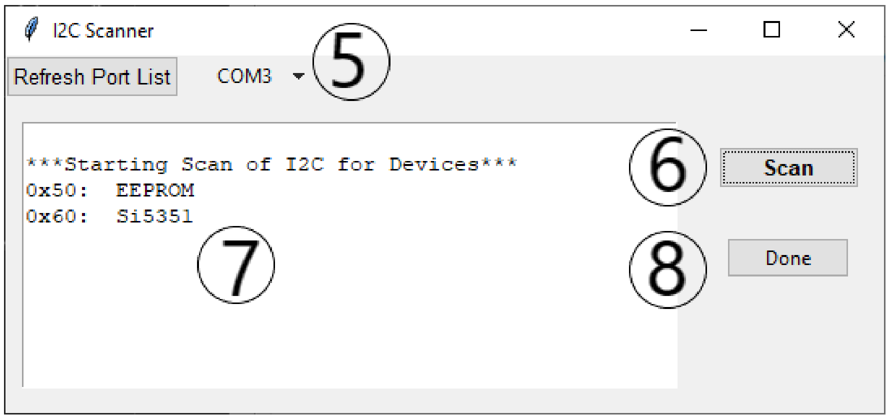
If you clicked the I2C Scanner button above, the window below would pop up. This allows you to scan your I2C bus.

5. Make sure you are connected to the right COM port. Hit "Refresh Port List" if you don't see your uBITX.

6. Click the "Scan" button. One round of scanning will be done of the 128 possible I2C addresses. You can click "Scan" a second time (or more) if you are concerned that a device has been missed for some reason.

7. The scanner will report the addresses of devices that it discovered. For COMmon devices, it will also tell you what it is likely to be. For example, in the above example, the external EEPROM (many modern processors do not have an onboard EEPROM, so an external one is required) is reported on address 0x50. The Si5351 clock chip is reported to be attached to its standard address at 0x60. Notice that no LCD was found. That was because the LCD on this uBITX was attached via the parallel interface!

8. And click Done when you are finished.



```
***Starting Scan of I2C for Devices***
0x50:   EEPROM
0x60:   Si5351
```

SDR

Dr. Lee spent quite a bit of his time investigating alternatives for connecting a SDR receiver (e.g., RTL-SDR USB dongle) to a uBITX. You can get a listing of his blog entries on this subject by going to this link:

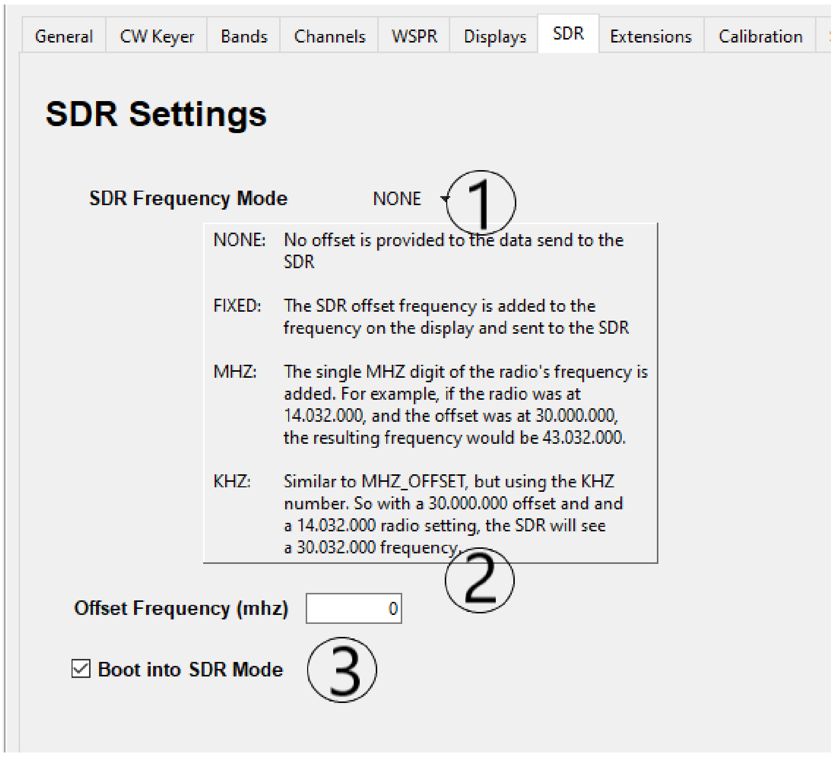
http://www.Hamskey.COM/search?q=sdr

This tab deals with the various configuration parameters that need to be set to support his circuit design to support a SDR attachment to the uBITX. Since this is a pretty involved hack (and there might be better and less invasive ways), most users will probably not need to set these variables. Indeed, most people's experience with the SDR feature of KD8CEC is actually negative. Somehow they click it, (look for "SDR" on your display) and suddenly they have no sound output! You want it to say "SPK" for speaker and not "SDR"!

By the Numbers

1. This drop down allows you to set whether you have a SDR attached and what type of frequency offset the SDR uses. The text below the "1" explains each option in detail. The MHZ and KHZ options are unusual, and I suspect that you will never need them.

2. This entry box is where you set the offset for the SDR. Each SDR has a standard offset, but often these are "nominal" and you need to adjust them for your particular SDR. Lots of tools out there for this.



SDR Settings

SDR Frequency Mode: NONE

NONE:	No offset is provided to the data send to the SDR
FIXED:	The SDR offset frequency is added to the frequency on the display and sent to the SDR
MHZ:	The single MHZ digit of the radio's frequency is added. For example, if the radio was at 14.032.000, and the offset was at 30.000.000, the resulting frequency would be 43.032.000.
KHZ:	Similar to MHZ_OFFSET, but using the KHZ number. So with a 30.000.000 offset and and a 14.032.000 radio setting, the SDR will see a 30.032.000 frequency.

Offset Frequency (mhz): 0

☑ Boot into SDR Mode

3. Normally, you put your radio into SDR mode and when you reboot (or power cycle) it returns to non-SDR mode (i.e. "SPK"). You can change this behavior if you want by checking the box. Then when you reboot, it will start up automatically in SDR mode.

Extensions

In addition to the various extensions to the original stock firmware of the uBITX, Dr. Lee invested a lot of effort in several major extensions that seem to have been forgotten. This section deals with two of them, Extended Keys and Custom Lowpass filters.

Many commercial rigs have provided a facility to have a remote "pod" where a user can tune the VFO and push one of several buttons to control the radio. This small device (the Elecraft K-Pod is perhaps the star here) allows the bulky radio to remain on the shelf and its control is replaces by something that is approximately 150mm square.

Extended Keys

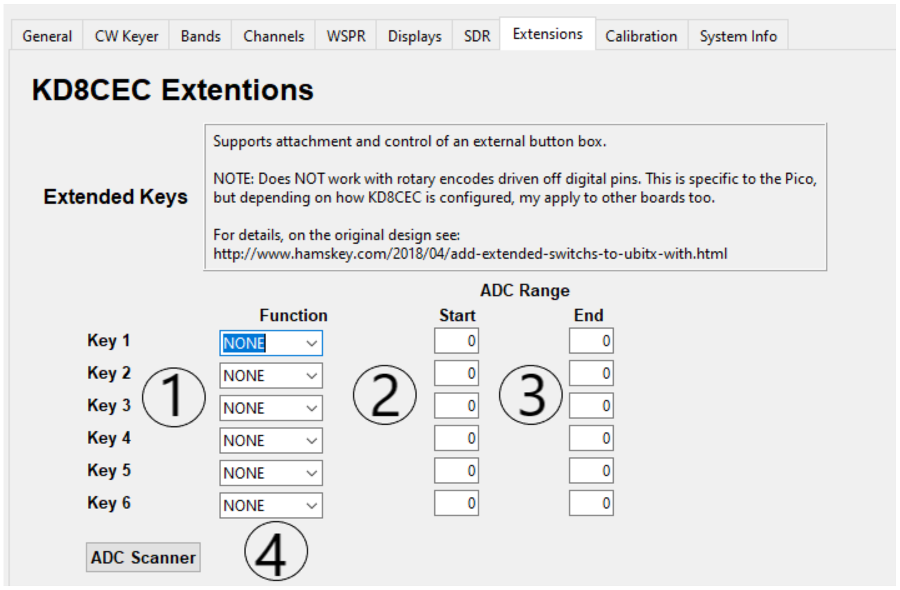
The Extended Keys function was Dr. Lee's effort to provide this functionality for the uBITX. The URL referenced in the text below will provide more technical details on how to implement it. The basics is that he re-used the encoder button interface (remember Nano pins are in short supply) and differentiate between the keys by the voltage level read at the analog pin used as input for the encoder button. For example, a voltage reading of close to zero is still assigned to a push of the encoder. However, if there was another button connected in parallel to the pullup that had a resister to ground, then a voltage reading somewhere less than 5 volts would indicate that the key was pressed.

So to get this to work, you need to create a set of buttons with a resister network, different resistance for each button and your encoder switch (the "push action") must be connected to an analog pin.

This means that encoders that are fully digital (such as in the current design of the PICO Raduino board will NOT WORK because the pins used do not read voltages.

By the Numbers

1. This menu allows you to select what happens when the key press is detected. There is a whole host of common functions you might like to invoke including Mode (change), Band Up, Band Down, Change the Tune Step, Switch VFO's etc.



KD8CEC Extentions

Extended Keys

Supports attachment and control of an external button box.

NOTE: Does NOT work with rotary encodes driven off digital pins. This is specific to the Pico, but depending on how KD8CEC is configured, my apply to other boards too.

For details, on the original design see:
http://www.hamskey.com/2018/04/add-extended-switchs-to-ubitx-with.html

	Function	ADC Range Start	ADC Range End
Key 1	NONE	0	0
Key 2	NONE	0	0
Key 3	NONE	0	0
Key 4	NONE	0	0
Key 5	NONE	0	0
Key 6	NONE	0	0

2. The starting ADC reading (0-1023) that will be trigger the recognition of this key.

3. The ending ADC value (0-1023, but greater than the start!) that will result in the key being triggered.

4. So how do you figure out what these ADC values are? Just click the ADC Scanner button and you can push buttons and see what the “safe” range is for a particular button. The explanation of the ADC Scanner will be reviewed in the next section “Calibration”.

Custom Low Pass Filters

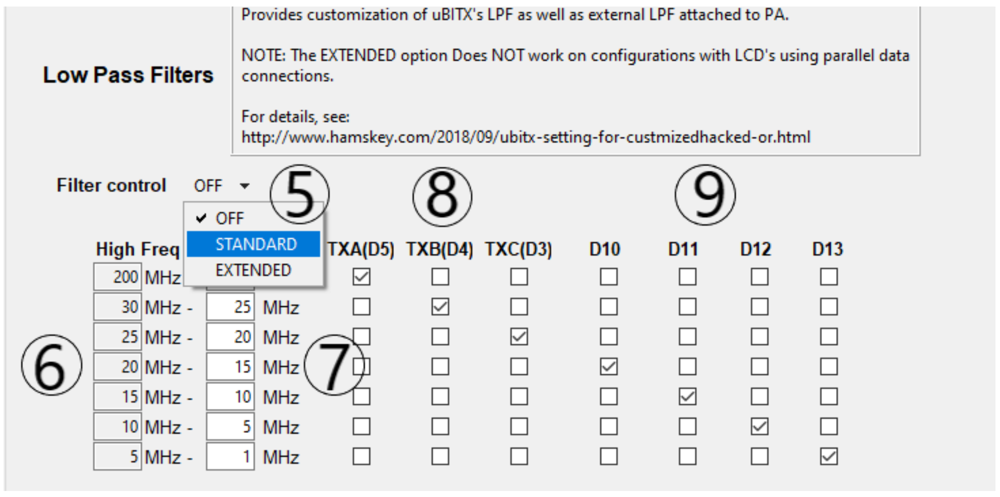
In the early uBITX v3/v4 days, there were significant concerns that the TX of the uBITX did not meet the standards set forward by the Federal Communications Commission (FCC) in the USA. There was a lot of experimenting in the community trying to figure out whether there was a combination of low pass filters that would make the uBITX complaint. I suspect that the original work by Dr. Lee in this area was primarily related to supporting this experimentation.

However, in solving this problem, Dr. Lee also provided a solution to signal other devices on the frequency bands the uBITX was using. These extra lines (repurposed from eliminating the parallel LCD) can be brought out and interpreted by external devices. I am not sure that was his original purpose or not, but it can be a very useful feature, even if you just use the standard filters and frequencies of the stock LPF network of the uBITX. The next screenshot reviews how to set these filters.

It is important to note that this is a compile time configuration parameter. The standard HEX files that were originally generated by Dr. Lee did **NOT ENABLE** this feature.

By the Numbers

5. This drop down allows you to turn off this feature, or select the “STANDARD” or “EXTENDED” set of lines. STANDARD lines are the ones already configured to the three relays built into the uBITX that control the LPF. EXTENDED adds to this the data lines that were reserved for a parallel LCD. These lines can be used to signal frequency settings to external devices as well as customization of a LPF.



Low Pass Filters

Provides customization of uBITX's LPF as well as external LPF attached to PA.

NOTE: The EXTENDED option Does NOT work on configurations with LCD's using parallel data connections.

For details, see:
http://www.hamskey.com/2018/09/ubitx-setting-for-custmizedhacked-or.html

Filter control OFF

High Freq		TXA(D5)	TXB(D4)	TXC(D3)	D10	D11	D12	D13
200 MHz		✓						
30 MHz -	25 MHz		✓					
25 MHz -	20 MHz			✓				
20 MHz -	15 MHz				✓			
15 MHz -	10 MHz					✓		
10 MHz -	5 MHz						✓	
5 MHz -	1 MHz							✓

6. Although you cannot directly enter this, these are the HIGH frequency limits. The values are just set to the start of the PRIOR band. So for example, the third band down as a HIGH frequency of 25mh. This was the LOW frequency of the prior LPF band.

7. This column is where you enter the LOW frequency of each set of filters.

8. For each pair of frequencies, you can select to enable or disable TXA, TXB and TXC. The referenced article by Dr. Lee provides you how these three lines are set by default.

9. These are the additional data lines that you can use when a LPF band is enabled. For example, you could use these lines to select an antenna or perhaps an external LPF on a PA.

IF Adjustments

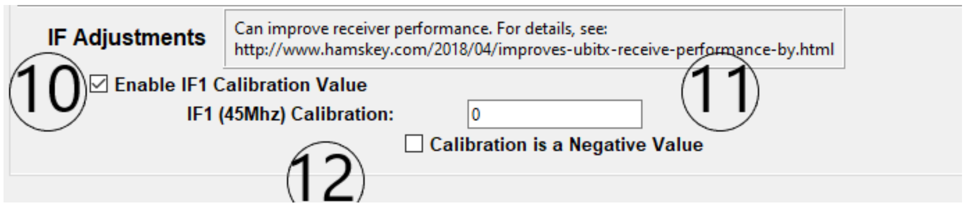
Dr. Lee discovered early on in his experimenting with the uBITX that tweaking the IF slightly can improve the reception of your receiver. I personally suspect that the benefit of this tweak has long since been incorporated in both the uBITX hardware and firmware. But judge for yourself. There is a link provided in the *SE*.

By The Numbers

10. This is just a checkbox to enable or disable this feature.

11. The article discusses how to figure out what should be entered here for the calibration figure.

12. If the calibration figure is negative, you need to check this box and enter the POSITIVE number in #11 above.



IF Adjustments

Can improve receiver performance. For details, see:
http://www.hamskey.com/2018/04/improves-ubitx-receive-performance-by.html

☑ Enable IF1 Calibration Value

IF1 (45Mhz) Calibration: 0

☐ Calibration is a Negative Value

Calibration

This is the largest set of numbers of any tab. However, for most users, these are the settings you DO NOT want to touch unless you are sure of what you are doing.

There are three general areas of calibrations:

- Radio frequency/BFO
- ADC associated with the press of a CW key
- S-Meter

Radio Frequency/BFO

Don't touch these except when you have no choice. Changing these values can result in not only your uBITX RX/TX on a vastly different frequency (Master Cal) than you think, but can also result in your radio being "deaf" to signals (BFO).

If you ever need to rediscover or tune these numbers, HF Signals has provided a nice video on youtube:

https://www.youtube.COM/watch?v=t6LGXhS4_O8&ab_channel=AshharFarhan

that uses a tuning aid hosted on their website:

https://www.hfsignals.COM/index.php/bfo-tuning-aid/

These resources are just for the V6. However, I have used them successful with my older uBITX.

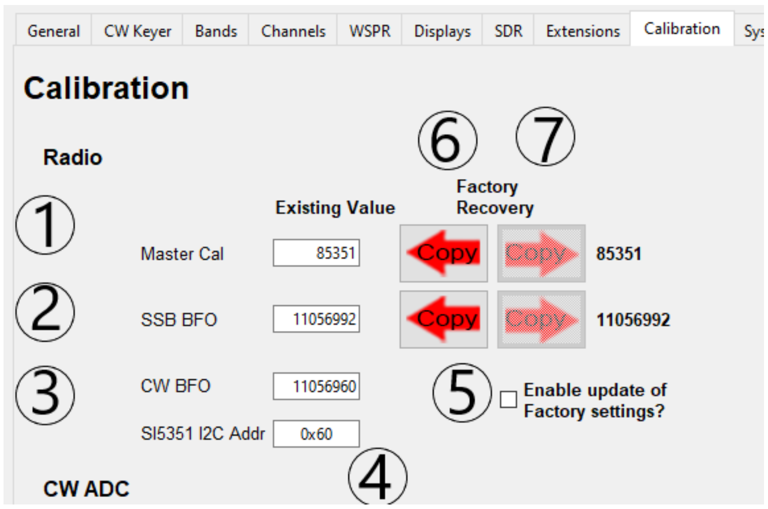
When you originally installs the KD8CEC firmware, it automatically copies the original settings for the Master Cal and SSB BFO to a special location in the EEPROM. This provides you with a "factory reset" in case you mess up these numbers and forgot what they were. The screen shot below shows the value for these factory numbers and, as will be discussed, allows you to restore or update these numbers.

By the Numbers

1. This is the Master Calibration number. Part of the mystery of this number is that it isn't just an "add this to the base frequency" number. Nor is it a percentage (we are off by .01%), Dr Lee has a special blog entry on just calculating this number. You can read it here:

   http://www.Hamskey.COM/2018/05/frequency-adjustment-of-ubitx-si5351.html

   Dr. Lee recommends calculating the new calibration and setting it using the uBITX Memory Manager (of course you can use the *SE* for that too). Reset the uBITX and check it again. I tend to favor the old school approach which is as done in the video above. Using the encoder, you can quickly change this setting and then check your results. (BTW: If you have a Nextion, this encoder based calibration is not turned on. So you have to follow Dr. Lee's guidance.



Calibration

Radio

	Existing Value	Factory Recovery
Master Cal	85351	85351
SSB BFO	11056992	11056992
CW BFO	11056960	
SI5351 I2C Addr	0x60	

☐ Enable update of Factory settings?

CW ADC

2. This is known as “THE BFO”. Dr. Lee created some confusion by by referring to this as the “USB Calibration” number. Since everybody seems to refer to this as the BFO, that is the terminology adopted here.

3. Interestingly, KD8CEC added the concept of a separate BFO for CW that did not exist in the original firmware. It is not clear whether this is essential differentiation as the (SSB) BFO and the CW BFO are generally very close. Although I am no expert on calibration, I ended up with both BFO numbers within about 30 points of each other.

4. This is the address of the Si5351 chip on the I2C bus (we saw it appear earlier in our discussion on the I2C Scanner app.) Like the rest of the calibration numbers, don’t touch unless you know what you are doing!
5. The “Factory” settings for the Master Cal and the BFO should not be changed unless you really know what you are doing. If you REALLY DO KNOW what you are doing, you can check this box and now you can change these settings too.

6. Clicking these left directional arrows will copy the Factory settings to the working settings of your radio.

7. These arrows are only made clickable if you have checked the checkbox mentioned in #5 above. Once you do, you can copy your existing settings over the Factory Settings. This means you Factory Reset will now be your current values.

CW Key Calibration

Calibrating your CW key is not as scarry as changing your Master Cal/BFO. But getting it right can still be a little tricky because the design of the CW key uses just one pin! For a straight key, the uBITX is ready to go. Hook it up and use the ADC scanner (see #12) to determine the ADC value when pressed vs not pressed. For example, on my rig, with a Nano, A6 reads 1023 without the key pressed. When pressed, A6 goes down to the low 300. So I would set the Start range to 0 and the End Range to 500.

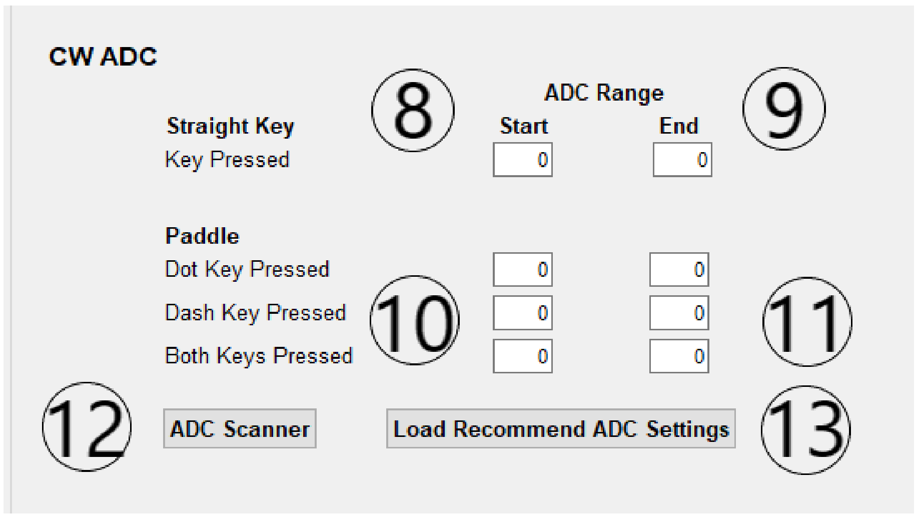
If you are using a paddle, you actually need to put together a small perf board with a resister network. This creates a create a different analog voltage on A6 when the Dot key or Dash Key or Both Keys are pressed. This article does a good job of expanding and describing this circuit:

https://ubitx.net/wiring-up-a-paddle-and-straight-key/

The *SE* (and the uBITX Memory Manager before it) allows you to set the bounds of the ADC for each key press as well as simultaneous presses of both keys.

By The Numbers

8. This is the beginning number for the ADC value when a straight key is pressed. Obviously, zero is the lowest it can be (but impossible to reach because of the 4.7k resister that is in series with A6.



9. This is the high-end ADC value for the key being pressed. On my stock V6 with a 4.7k resister in series, this number is around 325. You could enter 500 here and be pretty safe.

10. Assuming you have the resister network in place, you can check the value of ADC on each of the combinations of the keys pressed and see what the values are.

11. Same as 10 but you are looking for the value that is just higher than the ADC you see with the ADC scanner.

12. Click this to start the ADC scanner. This tool will help you a lot as you try to tune these ADC values.

13. You can also try loading the “Recommended Values” and then see which keys don’t work as expected. Then focus on tuning just those keys.

ADC Scanner

The ADC scanner can help you see the ADC values that processor is seeing when a key is pressed or the amplitude of the signal in the case of the S-Meter. It is important to note that this scan is that for a classic Nano. In the beta, pins that are not assigned (not available or used by the processor, will show a “0”. This limitation will be addresses before final release.

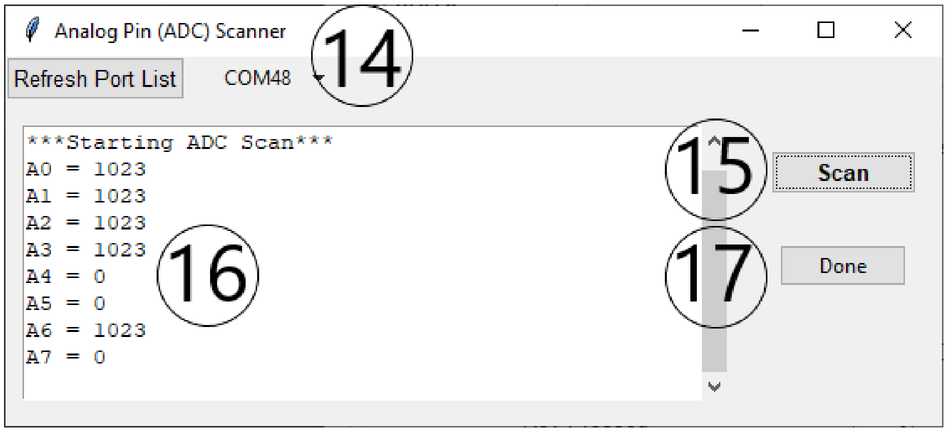
By the Numbers

14. As usual, make sure you are talking to the correct uBITX by checking on the selected COM port.

15. Click the Scan button to get a scan.

16. The ADC value of each of the Analog pins is reported in this section. If the values don’t look right, you can always click “Scan” one more time.

17. Click the Done button when you have the information you need to return to the primary UX of the SE.



```
***Starting ADC Scan***
A0 = 1023
A1 = 1023
A2 = 1023
A3 = 1023
A4 = 0
A5 = 0
A6 = 1023
A7 = 0
```

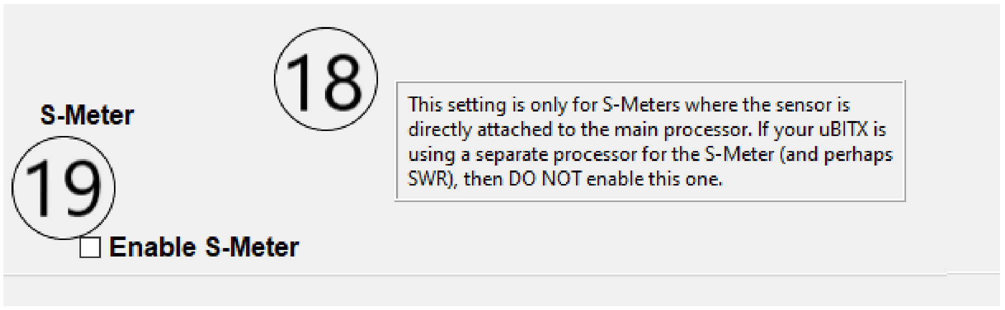
Configuring the S-Meter

You might not have to do anything here… These configuration parameters are for an S-Meter where the signal strength is directly being measured at the primary processor by a small (simple) sensor. See the following for the description of this sensor:

http://www.Hamskey.COM/2018/06/creating-simple-s-meter-sensor-for.html

By the Numbers

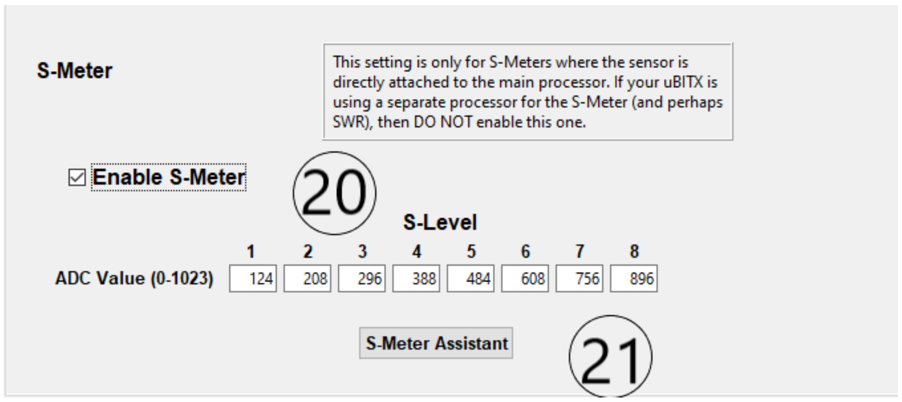
18. If you are measuring signal strength using a separate Nano, or you don't have a sensor at all, just skip this section.
19. If you are using a sensor like described in the link above (this is on board in my design), then click this checkbox.



This setting is only for S-Meters where the sensor is directly attached to the main processor. If your uBITX is using a separate processor for the S-Meter (and perhaps SWR), then DO NOT enable this one.

20. With the Checkbox clicked, you will have the option to individually set the ADC value for each S-level. For example, S-Level 7 is up through the ADC reading of 756.

21. Need some help deciding on the S-Levels? Then click the S-Meter Assistant.



S-Meter

This setting is only for S-Meters where the sensor is directly attached to the main processor. If your uBITX is using a separate processor for the S-Meter (and perhaps SWR), then DO NOT enable this one.

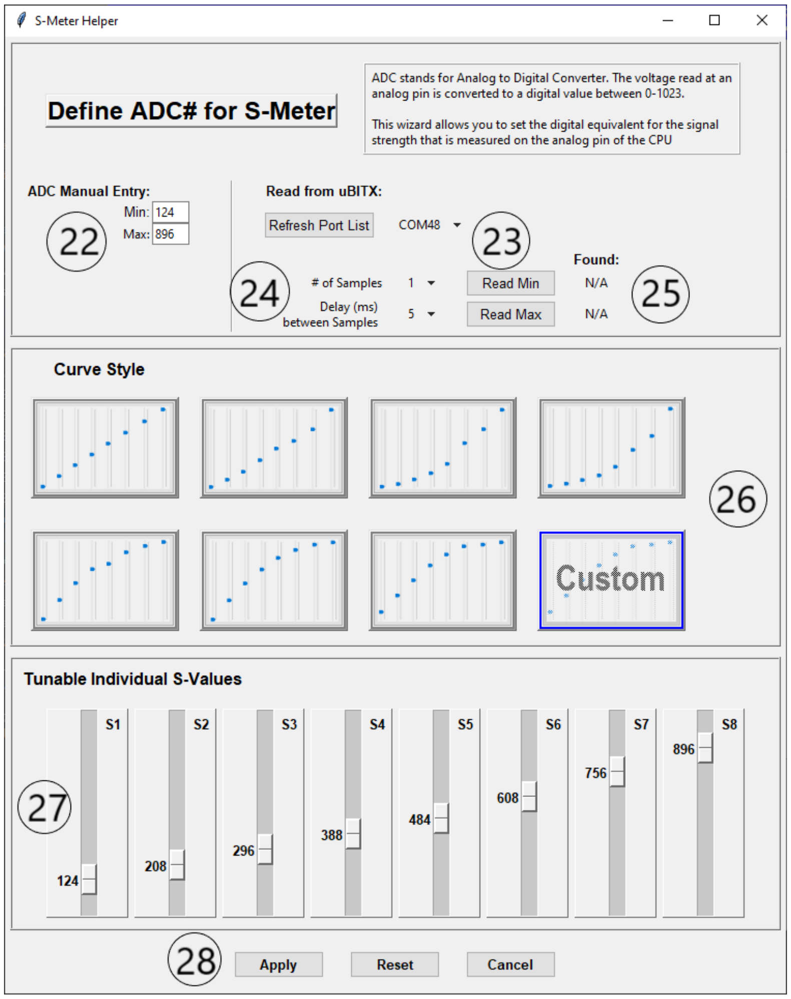
☑ Enable S-Meter

S-Level

	1	2	3	4	5	6	7	8
ADC Value (0-1023)	124	208	296	388	484	608	756	896

S-Meter Assistant

When you click the S-Meter Assistant button, the S-Meter Wizard appears. The goal of this wizard is to help you make a good initial choice for the ADC reading at each S-Level. Although you could try to exactly calibrate this, it is really a subjective feeling. One reasonable goal would be to try to get the uBITX S-Meter to be similar to the S-Meter of other radios in your shack.



S-Meter Helper

Define ADC# for S-Meter

ADC stands for Analog to Digital Converter. The voltage read at an analog pin is converted to a digital value between 0-1023.

This wizard allows you to set the digital equivalent for the signal strength that is measured on the analog pin of the CPU

ADC Manual Entry:
Min: 124
Max: 896

Read from uBITX:
COM48
# of Samples 1
Delay (ms) between Samples 5

Found:
Read Min N/A
Read Max N/A

Curve Style

Tunable Individual S-Values

S1	S2	S3	S4	S5	S6	S7	S8
124	208	296	388	484	608	756	896

The process of wizard is to designate a Low and High value for the ADC. Once you have that, you can then just select one of the sample curves provided and the rest of the S-Levels are adjusted accordingly. Then if you want to, you can tweak the individual S-Levels a little using the sliders near the bottom of the dialog.

By The Numbers

22. If you already have a good feeling for the lowest and highest ADC values your radio is hearing, then you can just enter these values hear to kick things off.

23. If you are not sure, you can scan the ADC to see what are the minimum and maximum values. First, make sure you are communicating with the desired uBITX by choosing the proper COM port here.

24. The two drop down menus allow you to decide how many samples you want to take (top menu) and the time between each sample (bottom menu). Although this may be useful, you need to be aware that zero values do happen a lot, so the “min” value will likely always be “0”. High end is a little better, but still will tend to be too low because of the result of min and max values being returned.

25. This area just reports the results of your scan. Note that these values are not automatically used. If you want them to be your new minimum and maximum, you need to type them into the boxes discussed in #22.

26. This section shows that given a minimum and maximum, how do you want the middle points distributed. There are 7 different distributions (by definition, what you have currently is a “Custom” distribution.) Click the icon of the graph you want your points to look like and the various intermediary points are distributed according to the graph.

27. You can tweak individual S-Levels using the scrollbars provided in this section.

28. Finally, click Apply if you like these values, Reset if you want to go back to the original values, and Cancel if you just want to quit with no changes.

System Info

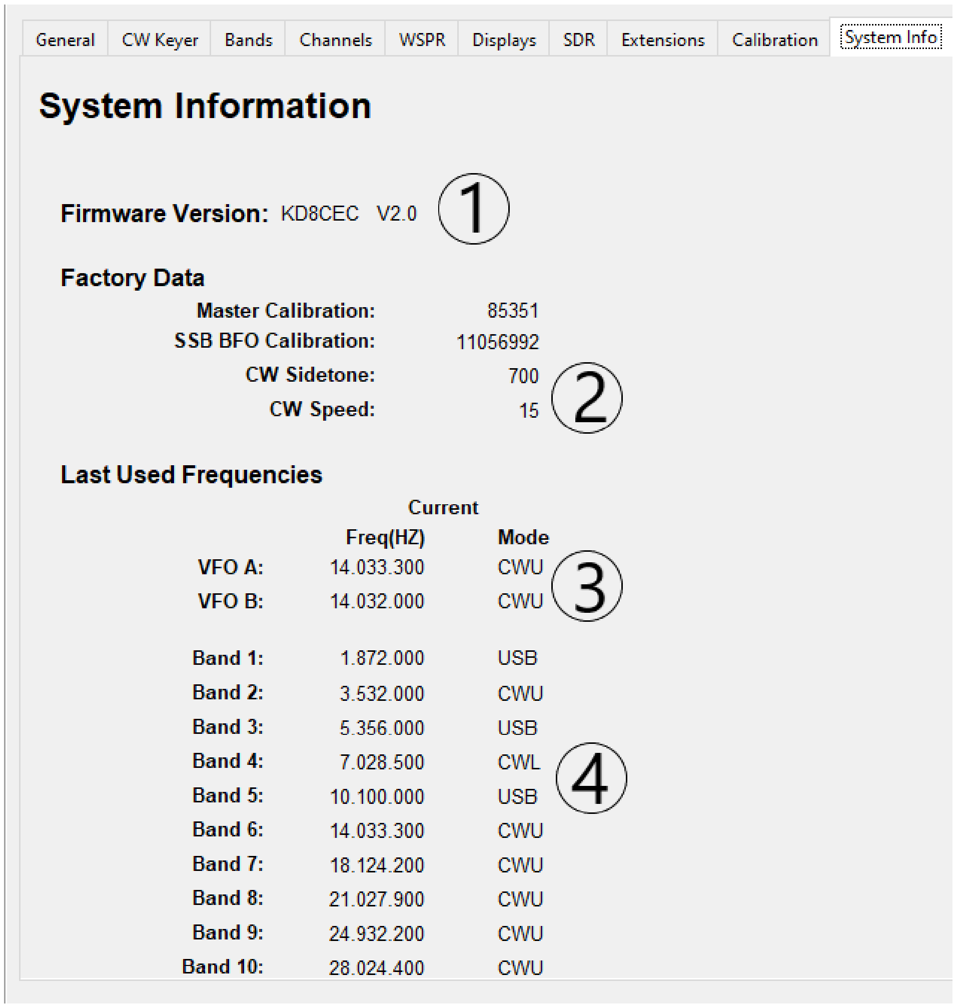
The final tab provides various information about your radio. Although basic at this point, the plan is to provide more information over time -- especially about which options have been selected and the purpose of various pins of the microprocessor.

By The Numbers

1. The actual version of the Firmware you are running on your radio is displayed here.

2. In addition to the Master Cal and BFO Factory Data, the Factory Data also records the original CW sidetone frequency and keyer speed. This information is shared here.

3. KD8CEC stores the last frequencies and mode for the VFOs (A and B) as well as the last frequencies used on each Ham band.



System Information

Firmware Version: KD8CEC V2.0

Factory Data

Master Calibration:	85351
SSB BFO Calibration:	11056992
CW Sidetone:	700
CW Speed:	15

Last Used Frequencies

	Current Freq(HZ)	Mode
VFO A:	14.033.300	CWU
VFO B:	14.032.000	CWU
Band 1:	1.872.000	USB
Band 2:	3.532.000	CWU
Band 3:	5.356.000	USB
Band 4:	7.028.500	CWL
Band 5:	10.100.000	USB
Band 6:	14.033.300	CWU
Band 7:	18.124.200	CWU
Band 8:	21.027.900	CWU
Band 9:	24.932.200	CWU
Band 10:	28.024.400	CWU

Final Thoughts

This *SE* is a work in progress and will probably always be that way. As new features are added within the KD8CEC firmware, the tuning of these features will require changes and enhancements to the *SE*. For a piece of software I never intended to write, it has certainly taken on a life of itself!

I look forward to your comments as you use the uBITX Settings Editor.

73

Mark
AJ6CU