Data loggers and Controller for the FrSky Imax B6 Mini Charger.

Introduction
The main python modules, imax_usb.py, imax_logger.py, and run_imax.py provide three versions to monitor and/or control the FrSky Imax B6 mini. The first two are Imax data loggers, using very different interfaces to see and log imax data in real-time. The third module is an advanced version that both logs and controls the Imax B6 mini.

All the work here was developed using an FrSky Imax B6 mini. Whether the applications will work with other Imax knock-offs is unknown. If a clone can use the FrSky Chargemaster software, then it will likely directly work with the applications here. See the last section of this document for more information on adapting non FrSky Imax chargers.

Examining the internet, there is additional confusion regarding the output ports of the Imax B6 and the Imax B6 mini. The former uses some sort of rs232/UART port and the latter uses a micro USB port as output. The applications here strictly apply to the FrSky Imax B6 mini.

The original intent was only to build a data logger, but by the time the lengthy process of deciphering most of the USB communications protocols had been unraveled, there was sufficient information to extend functionality to control the Imax.

The least complicated log module is a simple python logger program that prints data to the console and to a csv file using just python; the second logging version is driven from the default browser through a dashboard interface, and the final module offers a complete browser dashboard interface to both control and log Imax output.

The development of these applications is detailed in the last section of this document, *How were the Packet Sequences Obtained?,* along with tables defining the communication packet structure. The format of this last section takes a tutorial approach to how the communication data was extracted. Anyone wishing to extend, rewrite, or develop their own logging or control applications, especially a neophyte, will find the data in the last section useful as a starting point.

Application Methods

Introduction
The three Python based applications, are introduced in the order of their complexity. All three primary applications are bundled in one folder. At the very least, a rudimentary knowledge of python, and a knowledge of how to install additional external packages or libraries is important for understanding how to set up all these application.

Basic information and minimal python logging applications
Initially, several simple modules were developed as utility programs to test various aspects of communicating with the Imax B6 mini. *list_devices.py* and *imax_usb.py* contain a few methods that might be useful to scrape information from at least the Windows operating system to find the usb connection and configuration information. Most of the methods in these modules where found on the internet and modified only slightly. *list_devices.py* requires installing the hidapi package.

imax_usb.py is the simplest python logging program of the three primary application modules. It can also help locate usb device information. See the *main()* method for how to use the discovery methods.

The main function of *imax_usb.py,* was as a sandbox to test communications derived from observing the transfer of information between Chargemaster software and the Imax via the Wireshark packet sniffer.

To use the *imax_usb.py* module requires the python *pyusb* package to be installed first for usb communications.

imax_usb.py is a barebones logger. Follow these steps to use it (for Windows):

1. Plug in the Imax charger to your input power source, and connect the battery to be charged.
2. Configure your Imax charger settings as you normally would, but do not press the Start button yet.
3. Open a cmd or Powershell console window, and migrate to the folder containing the *imax_usb.py* module.
4. Assuming python is on Windows path, type: *python imax_usb.py*, but do not run the module yet.
5. Now press the ENTER/Start button on the Imax charger to start operation. If the buzzers are enabled you should hear the normal tones.
6. Now press enter in the console window to start *imax_usb.py.*
7. If everything is set correctly, you should see lines of data printed in the console.
8. To stop a run, press the Imax BATT/PROG/Stop button, or wait for the Imax to complete its function.
9. It will take two more cycles (around 20 sec. (if default data read interval is 10 sec) for the application to finish, and output the final data to *imaxdata.csv* file. The *.csv* file is overwritten with each new run.

This process must be run in the order specified, i.e., start the Imax first, then the app, or the program will error. Output consists of printing charging results directly to the console, and finally all the information to a *.csv* file. The module is not pretty, but It is functional.

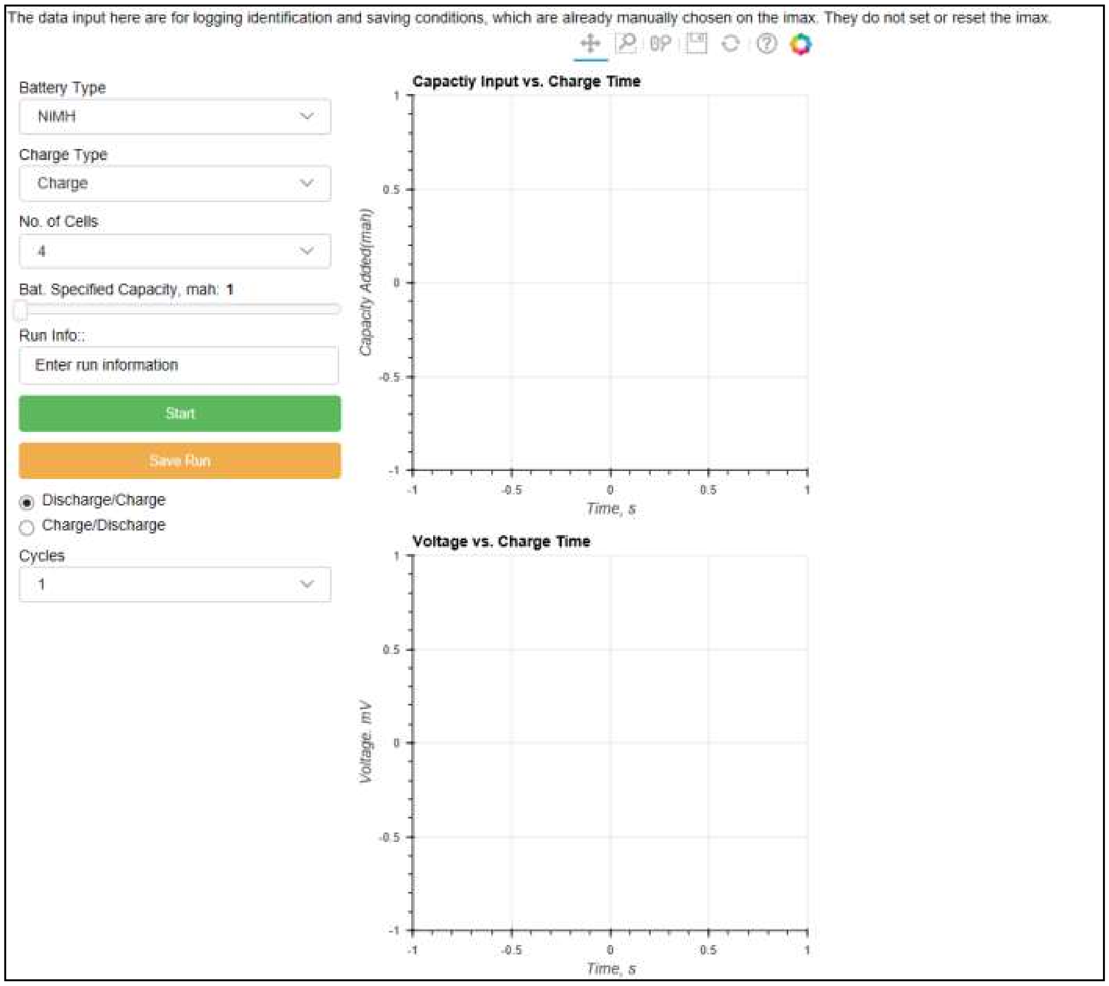
Although it is relatively straightforward to modify the code to allow starting operation in any order, this version does not have that feature. The second logger does.

Logger Application with Browser Interface

1. Browser interface app

The more user friendly application adds a browser interface to monitor the FrSky Imax B6 Mini charger, and outputs data to separate Excel *.xslx* files for each run, instead of to a *.csv* file. The application is known to work with the combination: python 3.6+, Windows 10, and Bokeh 3.2.2. IT'S BEHAVIOR WITH OTHER OPERATING SYSTEMS IS UNKNOWN. There are no specific calls to Windows, but all development and testing was done under Windows 10.

The interface is based on Bokeh, a python library for interactive visualization. It creates a server application using your default browser as a standalone application. Most of the browser page display is shown below. Not shown is a running log of events below the main parameters and plots.


The data input here are for logging identification and saving conditions, which are already manually chosen on the imax. They do not set or reset the imax.

Battery Type: NiMH

Charge Type: Charge

No. of Cells: 4

Bat. Specified Capacity, mah: 1

Run Info:: Enter run information

Start

Save Run

● Discharge/Charge
○ Charge/Discharge

Cycles: 1

To set up: Download all the files to a convenient folder. All output information will be created to this folder as well. In addition to a python install, the other packages or libraries to install are:

pyusb
bokeh
openpyxl
SQLAlchemy

The *pyusb* package is used for usb communications and *openpyxl* is used to generate the Excel or Open Office spreadsheet data storage files. Read the installation section for Bokeh, because it depends on several other packages and libraries. (*pip install bokeh*. Installation of *pyusb* was discussed in the first section above. The *openpyxl* package is used to generate an Excel *.xlsx* file as an output record.

The Imax logger charger app consists of two dependent python modules: *imax_0.py* and *imax_logger.py. imax_0.py* is similar to *imax_usb.py* previously described and is used to set the connection to the Imax and read some preliminary information, such as the limit settings. *imax_logger*.py contains the code to trigger data output from the Imax, and to generate the browser user interface.

To start operation, open an admin cmd or an admin Powershell console window. Migrate to the installation folder and type:

bokeh serve --show imax_logger.py.

This will start the bokeh server and read the set up file, *imax_logger.py*. If everything works correctly, your default browser will open as the image shows.

To stop operation from the console is a bit wonky. The bokeh server does not appear to have a graceful exit switch, so other than shutting down the console window, Ctrl-C is used to completely stop the server and application, but sometimes it has to be applied multiple times, or with intervening "Enter" key presses, and waiting a short time. For some reason, bokeh is a bit slow in responding to the Ctrl_C's.

The logging app does not control or change properties of the Imax, it only sends commands that request information from the charger as a data logger. Therefore, the charger must be set up using the Imax control buttons. Bidirectional control was not implemented to minimize potential safety issues. Bidirectional control is more involved and is available in the next application documented below.

2. General Appearance and Operation of *imax_logger*

There are three sections to the output screen. The first in the top left represents the conditions of the run. Filling in the data on the left is arbitrary, with one IMPORTANT EXCEPTION: If you are charging a Lithium battery type, but do not choose the type of lithium battery. the individual cell voltages lines will not be plotted. Internal charger operation is not affected, but the display will not reflect the state of all the cells. Most of the parameter data is based on dropdown lists that mirror the maximum or minimum settings allowed by the charger.

The "Run Info" text box allows the user to put in specific notes for further reference, if the data is saved. Again, remember that this data, does not affect or control charger operation, it is solely a guide to conditions for the user. The parameters settings on the left of the screen are only output when saving charger run data for future reference. If there is no interest in saving data for future reference; they can be left blank or at defaults.

On the right of the page are two real time plots displaying the battery charging or discharging status in real time. The top plot plots the capacity in milliamp - hours (mah) as a function of time. The lower plot plots the voltage in millivolts as a function of time (volts x1000); (the charger returns the millivolts as an integer). Below the left side input is a log of events; that show the current status or errors encountered during operation (not visible in the image). The user may need to scroll down to see the log, or by changing the browser zoom level to see everything at once. The log consists of two columns; a time stamp and the event. This log is not saved, only the data. Most events and errors are displayed here; keep an eye on it to see what is transpiring.

For best results, to start a run, follow this procedure:

1. Start the charger as you normally would, by plugging the charger into your input voltage source, and connecting to a battery.
2. On the Imax charger itself, set up the run conditions, using the charger buttons and screen. (This can be done any time before the charger's ENTER/Start button is pressed to start operation.)
3. Connect the Imax to the computer with an appropriate USB cable. Wait for the computer to acknowledge the connection.
4. Start an admin cmd or Powershell console window. Start the bokeh server as: *bokeh serve --show imax_logger.py*
5. When the browser page opens: Fill in the left side parameters, especially if the intent is to save the run, and/or, if you are charging a lithium type battery AND you wish to see the individual cell voltages, you must choose the battery type and the number of cells in the battery. Otherwise, just leave the defaults. (With the exception noted for lithium cells, the parameters can be filled in after the run is done, if you change your mind on saving.)
6. Click the "Start" button on the web browser screen.

7. Press and hold the "Start/Enter" button on the charger as you normally would to start operation. Remember that lithium batteries require a second confirmation button press. The app "Start" button will not do this.
8. The start button may change to "Connecting" and then to "Stop". This indicates the run has started, along with a event message being printed. The current timer settings update the graphs at 10 second intervals, so the first data point will appear in 10 s; wait for it. If the button does not change from connecting to stop, see the comments see the next section. Primarily, check all connections.
9. When the run is complete, and the Imax stops the charging operation, the "Stop" button will now show "Start". If you are using Imax factory defaults, the Imax will beep as it normally does.
10. The final values of capacity, voltage and charging time can be read in the log window.

The Imax charger itself and data acquisition can be stopped at any time by pressing the browser app "Stop" button. A run can also be stopped by pressing the Imax charger BATT/PROG/Stop button. The app will respond appropriately. If the charger has been set to audio alerts, you will hear an extra set of tones from the charger than usually heard. This has to do with the app requesting the final output values from the charger; it is not a charger problem.

3. Variant conditions on start up sequence: If the app "Start" button is clicked before the charger usb cable is connected, the Start button will show "Connecting". The user then has one minute to connect the charger (see the log table). If not connected in one minute, the Start button will display "Start", and the app will stop waiting for a connection and revert back to a non started state. Press the Start button to try again.

If the charger usb cable is connected, but the charger has not been started by pressing and holding the "ENTER/Start" button on the charger, the app will wait for one minute for the user to press the Imax ENTER/Start button. If that does not happen within a minute, the app will revert to a non start condition. Press the start button again to restart the process.

4. What to expect
As the battery is charged, the two plots will update automatically with the current capacity and voltage reported by the charger. Because data is acquired at 10 second intervals, the initial plots will have the appearance of a step function. As more data is acquired the plot lines will appear to "smooth" out. Also, because of the delay, on a rapidly increasing charge, the plotted data will not necessarily agree with the data on the Imax screen.

As stressed previously, if you intend to charge a lithium type battery, you need to both choose the battery type and the number of cells in the battery. the voltage plot will then show both the battery voltage and the individual cell voltages. In many cases, if the cells are in reasonable balance, the individual cells voltages will be superimposed on one another and not resolved.

As suggested by the preceding discussion, data acquisition will continue until one of three conditions occurs: The run is stopped by pressing the app "Stop" button; the Imax “BATT/PROG/Stop" button is pressed, or the charger has completed the requested operation.

The log table does not show any current data being collected, only the plots display the stats. However, raw data is printed in the console as the run proceeds.

When the run is complete or stopped by the user, the final values will appear in the log table.

5. Saving run data
Data may be saved after a run is done by clicking the "Save" button. The run will be saved in Excel *.xlsx* format as "imax_run_MM_DD_YYYY_HH_MM.xlsx" The data should be readable by Excel ver. 2007+ or Open Office 4+ (only tested with Excel 2007, but latest Open Office). The output will consist of the limit settings returned from the charger,

the data entered on the left screen. the final run values, and the complete real time data stream. The plots are not saved as images. Plotting data in Excel or Open Office is straightforward, and can easily be automated.

6. Starting another run
Despite the appearance of the Start button after a run is completed, the plots will not be reset if the Start button is clicked again. Any new data is added to the existing data. Moreover, there may be an unresolved issue with the application code itself and triggering the Imax. As far as the application is concerned, there are three ways to completely restart a run: If a stop was normally executed by you or the charger, try pushing the charger;s BATT/PROG/Stop button several times to bring the charger back to the top menu. Clicking a browser's refresh icon will completely refresh the page back to the starting conditions. The other way is to kill the bokeh server, using Ctrl-C a couple of times while in the cmd or powershell window, then restart as usual: *bokeh serve --show imax_logger.py)*. In addition, to either of these two methods, the Imax BATT/PROG/Stop button should be pressed. If the buzzers are engaged, you should hear three tones indicating the Imax has started, as well as the Imax screen indicating the desired status. No buzzer warnings usually means something is not triggering correctly. Usually the run will terminate in 20 s or so, or by pressing the Imax "Stop" button.

7. Internal Settings: Odds and Ends and Quirks
During development, sometimes clicking or pressing buttons inadvertently would cause the app to hang to the point that it would no longer respond to any commands. The exact point the code or the charger stopped caring was never certain. In the event it occurs, the only thing that works is to stop the app with Ctr-C's in the console window, disconnect the usb cable, and shut off the charger. Basically, do a clean restart.

Data is accumulated at a fixed rate, currently set to 10 seconds. This can be changed in the *imax_logger.py* code: Find the *start_device()* method and change the last value in the line, e.g., *start_cycle = curdoc().add_periodic_callback(update, 10000)* to any value in milliseconds you wish. At a data rate of 10 s, data will be accumulated at the rate of 360 points/hr. For most purposes, the rate is likely sufficient. The lower limit of data collection time has not been determined, but a safe guess would be to keep the rate at above 300ms to allow for data collection and processing, especially if the computer is being used for other purposes during data collection. (It is likely the minimum turnaround time can be reduced, by streamlining some of the code.) Battery capacity is not calculated by the application, it is read from the Imax charger, so there are no issues with capacity integration and resoltuion, if the data acquisition time is changed.

Similarly, the grace periods on waiting for the device to be connected, or for the Imax ENTER/Start button to be pressed can be changed. There are two time values that control each of these processes. Device searching is controlled from the *imax_logger.start_device()* method, using the *datetime.timedelta* function to set the maximum length of time to wait for connection (currently 1 min.), and a looping conditional, using the *time.sleep()* function set to 1 second. Similarly, the wait cycle times, if the charger has not been started, can be found in the *button_startstop handler()* method.

In addition to the annoyance of trying to stop the bokeh server in the console, with at least Windows, one other annoyance was found, but only when using bokeh version 12.15. The bokeh server and Windows 10 had some sort of packet communication glitch that does not affect operation, but does keep printing a warning message every ~10-15 seconds to the console. The warning does not affect operation, but the message incessantly appears in the console when the server is idling. This issue was fixed in bokeh version 12.16.

Advanced Imax Application - Control and Storage

1. Introduction

As already explained, the logger app version does not change Imax settings, only reports them. Because of the very limited scope of the host requests needed to read out all the Imax conditions, the simple logging code is considered safer.

Within the scope of the packet communication deciphering, much of the data was already available to decipher the packet format sent from the host to change in the Imax run conditions. Because of the more complicated handling of information, the details of this information to create a control application are described separately in the last section of this document.

The interface used is again based on your default browser with a dashboard type format. The interface is generated using the python based bokeh package.

This version expands on the simple logging version:

- Control of the Imax B6 Mini is driven from the dashboard. With the exception noted below about limit settings, the most common functions are set and transferred to the Imax and a run started or stopped using only the dashboard.
- When a battery is picked, the values initially displayed for charging or discharging are considered safe charging values.
- As with the logging application, real-time data is displayed graphically.
- The application uses a database to store commonly used settings sets for different or the same battery. The number of programs that can be saved is only limited by available storage space.
- As with the logging application, run data can be archived as Excel files, if desired.

Limit settings, i.e., max input voltage, cycle delay time, max charge time, or buzzer settings are not dealt with in the application. These must be manually set through the imax itself. The application will record the values of these settings, but does not change them.

A number of the comments about operation described in the logging application also apply here. To avoid excessive repetition, please read the logging application instructions to understand some of the quirks.

Five files are needed to run the Imax Controller version:

- *run_imax.py* is the main file,
- *imaxconfig.ini* allows some settings to be personalized when the app opens.
- *imax.py* contains most of the specific methods for setting up communications.
- *db_ops.py* contains the methods for managing the database.
- *tabledef.py* creates a new database for storing programs.
- *imax_programs.db*, sqlite database file to store imax settings as programs.

2. Setup

Besides ensuring the four files are in the operating folder, python will need all the dependent libraries and packages previously mentioned for the simple logger application: *pyusb, openpyxl, bokeh, SQLAlchemy*.

A configuration file*, imaxconfig.ini* contains some initial settings that can be changed to accommodate more common user-based settings when the application starts. Not all initial settings are changed from the *.ini* file. Only the battery type, number of cells, maximum nominal capacity of the capacity slider, and initial starting capacity are represented. Other settings are defined at the beginning of the main control module, *run_imax.py*. Care should be used in changing these values. For instance, the program does not accept charge rates over the nominal battery capacity. (To do this, the code itself would have to be changed.) This limitation was a safety feature, based on what is generally considered good practice in the literature.

To start the Imax controller application, from a cmd or powershell window do:

bokeh serve --show run_imax.py.

As with the logging version, this will open the application in your default browser.

3. Imax Control

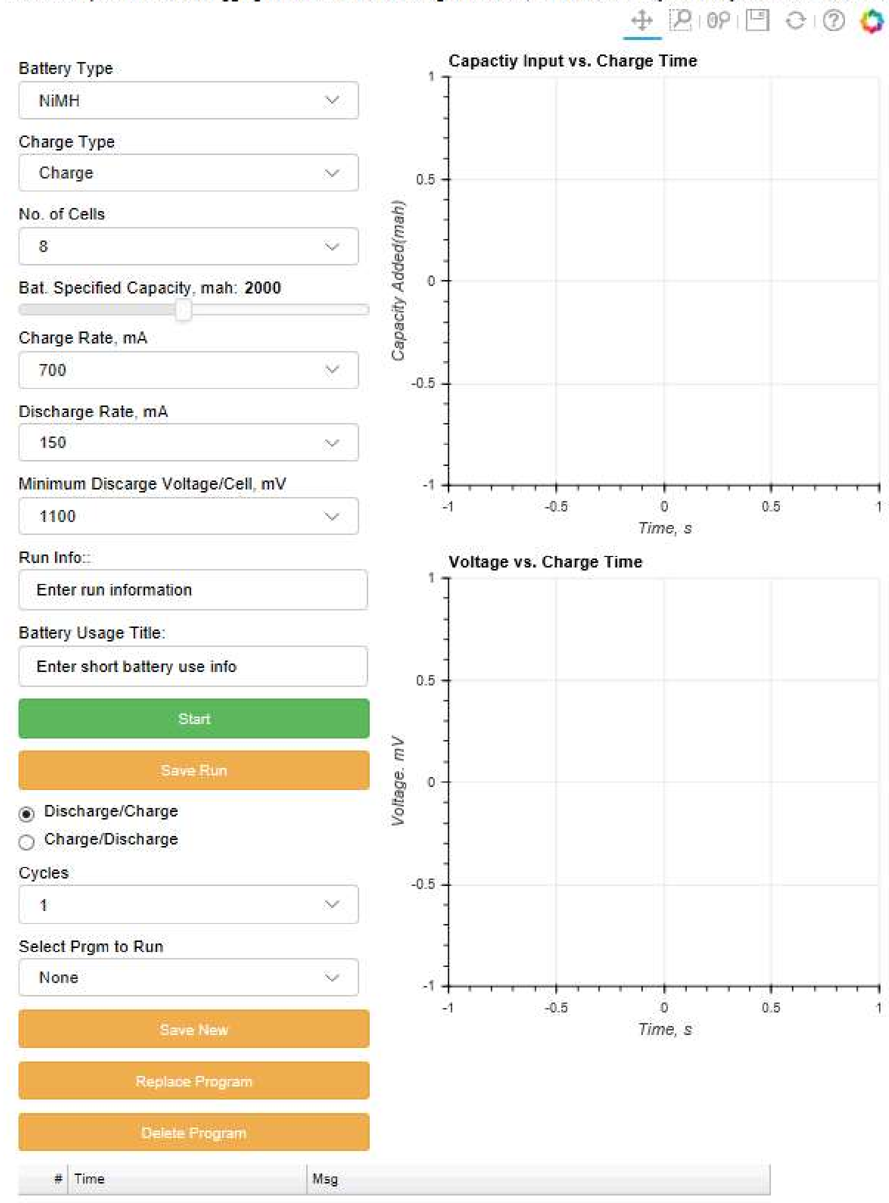
Most of the comments on the logging application regarding initializing the connection and operation apply here as well. For completeness' sake, the relevant start up information is repeated here:

1. Start the charger as you normally would: plugging in the charger to an input voltage source, and plug in the battery to be charge/discharge.
2. Connect the Imax to the computer with an appropriate USB cable. Wait for the computer to acknowledge the connection.
3. Click the Imax BATT/PROG/Stop button a couple of times to put it in proper standby mode.
4. Start an admin cmd or Powershell console window. Migrate to the folder containing the application.
5. Start the bokeh server as: *bokeh serve --show run_imax.py*
6. When the browser page opens: Either fill in the left side parameters, or choose a program to run. (Details in the next section.)
7. Click the "Start" button on the web browser screen.
8. The start button should rapidly change to "Connecting" and then to "Stop". This indicates the run has started, along with a event message being printed. The current timer settings update the graphs at 10 second intervals, so the first data point will appear in 10 s; wait for it. If the buzzers have been enabled, there will be three tones. If no tones or only one tone is heard. it is likely something has gone wrong. Click the application stop button. Press the BATT/PROG/Stop button again, press the browser recycle ion, and retry.
9. When the run is complete, and the Imax stops the charging operation, the "Stop" button will now show "Start". If you are using Imax factory defaults, the Imax will beep as it normally does.
10. The final values of capacity, voltage and charging time can be read in the log window.

4. Control Interface

The Imax controller software version has a similar, but expanded input interface compared to the simple logging version:

The data input here are for logging identification and saving conditions, which are already manually chosen on the imax. They do not set or reset the imax.


Battery Type: NiMH

Charge Type: Charge

No. of Cells: 8

Bat. Specified Capacity, mah: **2000**

Charge Rate, mA: 700

Discharge Rate, mA: 150

Minimum Discarge Voltage/Cell, mV: 1100

Run Info:: Enter run information

Battery Usage Title: Enter short battery use info

[✓ Discharge/Charge] Charge/Discharge

Cycles: 1

Select Prgm to Run: None

#	Time	Msg

Now the selection controls, text boxes, and radio buttons actively control the Imax charger, and need to all be considered before starting a charge/discharge run. There are controls to save and retrieve Imax settings as a 'program': An additional dropdown list allows user saved programs to be retrieved; there is a “Battery Use" text box, and buttons to save, replace or delete programs from a database. The plots are the same. Most of the comments on operation or quirks in the application are the same as noted above in the simpler logging module.

The nominal battery capacity slider range is set to 4000 mah. This maximum can be changed in the imaxconfig.ini file using any text editor. For very larger capacity batteries, it may be difficult to lock on the nominal value. The best technique is to get close to the value, hold down the LMB on the slider button and use the keyboard arrow keys to step to the desired value.

The charger can be run by manually altering the settings and clicking the Start button as with the logging application, or by retrieving a saved program containing commonly used settings for a particular battery. The difference between the logging application and the control application is that the new interface drives the Imax operation, as well as interprets information sent from the Imax.

Manually changing settings can affect other settings. Changing the battery type will automatically change several other settings. For instance, changing the nominal battery capacity (manufacturer specified battery capacity) slider will change the charge rate and the charge rate dropdown range. The new charge rate overrides any previous value, and represents the safe charge rate for the type of battery and the nominal battery capacity. This is a safety feature. The charge rate can be changed by changing the charge rate *after* changing the nominal capacity.

As with the logger application, for very larger capacity batteries, it may be difficult to lock on the nominal slider value. The best technique is to get close to the value, hold down the LMB on the slider button and use the keyboard arrow keys to step to the desired value.

Settings retrieved from a program can be manually overridden at any time before starting a charging run. Of course, any manually changed settings will not be retained, unless saved over an old version of the program settings, or saved as a new version. As a good practice, to avoid other values changing unexpectedly, make changes starting at the top of the settings group, working your way down. **Settings changes made after a run has been initiated are not executed. Stop the Charger, make any changes and restart the Charger.**

5. Saving and retrieving saved settings (programs)

An especially useful feature is the ability to save commonly used Imax settings for the same or different batteries. The Battery Use text box must be filled in to provide a mnemonic to both organize and retrieve different program settings for different batteries, or different settings for the same battery. How a battery is used is likely the easiest way to organize your groups of settings. The text in the 'Battery Use" text box should be kept short, like a title (in contrast to the Battery Info text box) with sufficient information to identify the charge conditions for the battery's intended use relative to other stored settings. For example, "Sukhoi RX NiMH, 1450 mah, chrg: safe C" might be a sufficient description for charging a NiMH receiver battery pack at 0.4 C in a Suhkoi model. The database is managed according to the description, so avoid duplicate descriptions. If there are duplicates, the application will only choose the first version it retrieves, which is likely to be the earliest version. If the user neglects to provide any Battery Use text, the application will create a unique text, based on several settings. The text will be unique, but descriptively general and cryptic. *There is one caveat to choosing programmed settings: If a prgm is chosen, and you manually make changes to the settings, then decide to go back to the original settings, you CANNOT just click on the same program in the program list to return the settings. You must FIRST click on another program set, then click again on the desired program.* This has to do with the way bokeh handles a selection event.

Any of the program settings retrieved from the database can be manually modified at the user's discretion. The changed values are not automatically saved to the database.

Only Imax settings and the battery usage text are saved to the database. Text in the "Run Information" text box is considered specific to a run, and is only saved using the separate "Save Run Data" button. Settings related to the usb connection, are also not saved by any of the output methods. These values are determined when the "Start" button is pressed, and might be dependent on any operating system quirks.

Saving or deleting programs is independent of whether the Imax is connected or not connected to the host. Only when the Start button is pressed, will the battery specific settings be transferred to the Imax, not before. ***Changing settings, saving, deleting, or replacing programs must not be done during the run.*** The Imax charger will continue to run under the old conditions, but data logging will be compromised.

Initially, the database file, *imax_programs.db* contains several example program sets as a starting point for exploring its use. This file can be complete removed and rebuilt, if necessary. To reset, delete the original file from the folder. After opening a cmd or Powershell windows in the folder containing the installed module files, execute *python* *tabledef.py* (assuming python is accessible globally through your Windows Environment Variables PATH). This module will regenerate an empty database file with a single table named, 'programs' of class Programs.

6. Saving runs
Run data can be saved for future retrieval and comparisons as with the logging version. As already stated, saving run time data is independent of saving the program settings. Run data is saved as an Excel file, as discussed in the logger application.

7. Starting another run
There is no simple way to clear data and run again. To start a new run, assuming no internal program error, i.e. if a stop was normally executed by you or the charger, push the charger's BATT/PROG/Stop button at least three times to bring the charger back to the top menu, then click the browser recycle icon. This should reset the dashboard and you should then be a able to pick your conditions or a program and restart.

If you have the buzzers on (factory default) then you will hear three different tones, if the charger and application are working together, If you hear nothing, or one tone, it usually means something went wrong somewhere. Wait for the operation to suspend itself in around 20 seconds, then press the Imax start button three times and retry. (I was never quite able to figure out all the reasons for why it would fail at times.)

8. Additional operation notes
The application starts with nominal values and parameter ranges set for a single cell NiMH battery. If manually changing settings is mostly used, these two default values can be permanently changed by setting the battery type variable 'bat_type' and 'cells' in the *imaxconfig.ini* file, using any text editor.

Changing the battery type and especially the nominal battery capacity slider will automatically change the charge rates, and minimum voltage per cell. The current rate is set to 40% of the maximum mah capacity as a conservative charge rate for nickel batteries, 50% for lithium batteries, and 30% for lead/acid batteries. These are considered "safe C rates", but can be changed using the dropdown lists. The charge rate range is limited to a maximum of 1C.

The discharge rate list is set to a maximum of 2000 mA. This is the same range as allowed by the Imax charger.

To delete a program, it must be first recalled using the program selection dropdown list. The parameters appearing in the various controls will still be displayed, and will only change whenl a new program is chosen, or the individual parameters have been manually changed.

The Imax charger has no cycle operation for Lithium type batteries, or Pb batteries. This is reflected in the application with cylices set to zero. Any attempt to change the value will always return to zero. Also, a program will automatically save the number of cycles as zero unless the charge type is set the 'Cycle'.

Cycling NiXX batteries show a continuous time cycle. This is not directly the way the Imax outputs the information, it resets the time index to zero at each stage of a cycle (two times). The application adjusts for this and shows the time in a continuous stream. However, the capacity curves will jump dramatically in values as the cycle shifts, because the Imax resets the capacity back to zero for the second part of each cycle. Keep in mind there are 3 phases to each cycle, C/cycle delay/D or the reverse. These three phases will be obvious from the way the data plots.

How were the Packet Sequences Obtained?

The information here is somewhat of a "how to" tutorial to communicate and extract usb information between devices, as well as providing a reasonably comprehensive summary of the Imax B6 Mini usb communications.

An admission: I am less than a neophyte understanding the ins and outs (yes, a pun) of USB protocols. I did learn a few things along the way, and picked up a smattering of the jargon, and figured out how to see the communication stream between a computer and the Imax B6 mini, but my "knowledge" remains shallow. Ask me anything deeply technical about the intricacies of ports and USB protocols and I will fold into a knot. This helped me understand the USB structure a little: http://www.usbmadesimple.co.uk/ums_4.htm

1. FrSky Imax B6 Mini Operation.
Examining the internet, there is confusion regarding the output ports of the Imax B6 and the Imax B6 mini. The former uses some sort of rs232/UART port and the latter uses a micro USB port (which is really a USB/UART interface board). The output information available on the internet for the *Imax B6* IS NOT RELEVENT for the *Imax B6 mini*. Hence, the reason for this work. When the term Imax is used here, it strictly refers to the FrSky Imax B6 mini.

Whether the information is applicable to knock-off versions of the FrSky charger that contain a usb port is not known. Naively, two factors can dominate differences: One, the charger must have a usb port for output, and second, it will depend on the USB/UART microcontroller board used in the charger. If the clone has a USB port, and can use the Chargemaster 2.03 FrSky software, then the modules here will likely work. If the latter is not true, then this entire section is especially for you. *[Although it is certainly possible to generalize the software here to handle multiple Imax knock-off charger versions by incorporating a standardized packet template file system, the current version is not structured in a fashion to be trivially recoded. However, the way the application is set up, the changes to generalize would be relatively straightforward.]*

All the effort discussed here was under the Windows 10 operating system. Much of the work and modules will apply with no or minor modifications to other operating systems, provided the appropriate python os modules have been installed. Python version 3.6.1 was used for the development, although it is expected the code will work directly with python 3.x.x versions.

2. Chargemaster IMAX B6 Mini byte sequences Information.
With perseverance, I obtained and deciphered most of the packet sequences used to communicate and control the Imax B6 Mini. Regarding the python coding, the code by Milek7 provided me with an excellent starting base for understanding how some of the USB packet information was arranged.

For those who might also be USB communication blind, I explain in detail the process of how I deciphered the information. The format is a cursory tutorial on the communication discovery process. I realize to some users and developers, this level of detail may be annoying, but I often find background information is too sparse for the occasional coder, hobbyist, or non professional to really understand a process. My style is different. Of course, some level of technical knowledge is a must. In this case, at least a rudimentary knowledge of the Python script language, is necessary to understand most of the discussion, and use the applications.

First a USB acronym: HID stands for Human Interface Device. On at least a Windows 10 system, this is the section title under Window's Device Manager where the Imax charger will show up when a USB cable connects the two devices.

The first task was to find the Imax device on Windows and obtain the device (Imax) usb configuration information. As it turns out, some of the information was already available in the module from Milek7, but the consistency and origin of the information was not clear.

3. Installing necessary Python Modules and Libraries
Two python modules that appear to be in common use for usb communication were *hidapi* and especially *pyusb*. Despite a number of attempts, and for reasons beyond my abilities, I was unable to get the *hidapi* library to read data from the Imax in my Windows 10 system, although I was able to use it to list devices. Pyusb has the advantage of

being a bit easier to understand than hidapi. The latter is supposed to be os independent, but suffers from a lack of python examples, especially for the Windows os. Milek7 used hidapi, but not with a Windows system. However, the pyusb package worked flawlessly, and was used for the logger and controller scripts.

Installation of pyusb requires libusb-1.0.dll or greater be installed first. The library is a lower level, compiled C++ library for interfacing between various languages and operating systems. However, with 64 bit Windows 10, the installation was a bit strange, and I searched quite a few internet posts before finding how to install it on Windows, so I will describe the installation: Download the appropriate libusb-1.0 binary archive files; they can be downloaded as zip or 7z. Open the archive. There are two subdirectories in the archive: for 64 bit or 32 bit windows systems. Migrate to the 64 folder containing the already generated libusb-1.0 files. Copy the three libusb.-1.0 files, including the .dll to both C:/windows/system32, and for Windows 64 bit systems, also copy to C:/windows/syswow64. There is an alternate method for using the libusb files; see the main data logging script file, *imax_usb.py* for information.

4. Getting at the USB Information and Configuration
Thanks to many people, the behind the scenes communications between the os and the device are automatically taken care of, so we can concentrate only on the immediate communications between the host system and the device.

When connected to an operating system, a USB device identifies itself by a VID/PID combination, which is assigned to the device by the manufacturer. A VID is a 16-bit vendor number (Vendor ID). A PID is a 16-bit product number (Product ID). The computer uses this VID/PID combination to find the specific or general drivers to use for the USB device. To understand how a device manufacturer has configured the USB configuration and communication protocols for the device, we need to know the VID and PID to communicate with the device. When the Imax was connected to the computer usb port the first time, a notice flashed up about a development board device general driver being installed. The information disappeared too quickly to be useful. A small file, *list_devices.py* was adapted from a couple of sources to read the hid device information from windows using the hidapi module. Running the module in a command window as usual: *python list_devices.py*, flashed a large list of devices of all types. As it turned out, the last one was the Imax device, but that may not always be true, considering the plethora of USB/UART devices. *list_devices.py* also creates a csv file, *devices.csv*, in the same directory as the python script as permanent output. The Imax device entry was:

```
usage_page : 1
usage : 2
interface_number : 2
path : b'\\\\?\\hid#vid_0000&pid_0001#8&1ecab2b5&0&0000#{4d1e55b2-f16f-11cf-88cb-001111000030}'
vendor_id : 0
product_id : 1
serial_number :
release_number : 0
manufacturer_string : SILICON LABORATORIES
product_string : C8051F3xx Development Board
```

Removing the usb cable, and re-running *list_devices.py* showed no entry for this device, so this information was for the Imax. Another simple way to find the device information is to check the Device manager HID section before plugging in the device and again after plugging in the device, and find what was added, but you may need a good memory to remember what was there and is now missing.

With the above information, in Device Manager, under HID - compliant device for the Imax, the following important pieces of information could be confirmed, or gleaned: *Device HID\VID_0000&PID_0001\....;* this information tells us the two hexadecimal codes are: vid = 0x0000 and pid =0x0001.

The VID and PID turn out to be the same as indicated in the code from Milek7. The next stage of the communication breakdown process was to determine just what the control, configuration, data request, and output packets looked like. There are several ways to get at this information. One was Wireshark, a software communication packet sniffer. However, initially, I found python methods that would get the information. The small script module, *find_imax.py*, was put together to specifically get the device configuration using the vendor and product id's found above. The data is printed to the console. The output was:

```
Decimal VendorID=0x0 & ProductID=0x1

device found:  DEVICE ID 0000:0001 on Bus 002 Address 012 =================
 bLength              :  0x12 (18 bytes)
 bDescriptorType       :   0x1 Device
 bcdUSB               :  0x110 USB 1.1
 bDeviceClass          :   0x0 Specified at interface
 bDeviceSubClass       :   0x0
 bDeviceProtocol       :   0x0
 bMaxPacketSize0       :  0x40 (64 bytes)
 idVendor             : 0x0000
 idProduct            : 0x0001
 bcdDevice            :   0x0 Device 0.0
 iManufacturer         :   0x1 SILICON LABORATORIES
 iProduct             :   0x2 C8051F3xx Development Board
 iSerialNumber         :   0x0
 bNumConfigurations    :   0x1
  CONFIGURATION 1: 64 mA ==================================
   bLength             :   0x9 (9 bytes)
   bDescriptorType     :   0x2 Configuration
   wTotalLength        :  0x29 (41 bytes)
   bNumInterfaces      :   0x1
   bConfigurationValue  :   0x1
   iConfiguration      :   0x0
   bmAttributes        :  0x80 Bus Powered
   bMaxPower           :  0x20 (64 mA)
    INTERFACE 0: Human Interface Device ====================
     bLength           :   0x9 (9 bytes)
     bDescriptorType   :   0x4 Interface
     bInterfaceNumber   :   0x0
     bAlternateSetting  :   0x0
     bNumEndpoints      :   0x2
     bInterfaceClass    :   0x3 Human Interface Device
     bInterfaceSubClass :   0x1
     bInterfaceProtocol :   0x2
     iInterface         :   0x0
      ENDPOINT 0x81: Interrupt IN ==========================
       bLength          :   0x7 (7 bytes)
       bDescriptorType  :   0x5 Endpoint
       bEndpointAddress :   0x81 IN
       bmAttributes     :   0x3 Interrupt
       wMaxPacketSize   :  0x40 (64 bytes)
       bInterval        :   0xa
      ENDPOINT 0x1: Interrupt OUT ==========================
       bLength          :   0x7 (7 bytes)
       bDescriptorType  :   0x5 Endpoint
       bEndpointAddress :    0x1 OUT
       bmAttributes     :   0x3 Interrupt
       wMaxPacketSize   :  0x40 (64 bytes)
       bInterval        :   0xa
```

This is a lot of useful detail: Much of it goes beyond my investment in understanding USB protocols. Apparently, the Imax usb configuration protocol is fairly simple, compared to some devices. Some of the more critical information is under the ENDPOINT headings. An ENDPOINT, a term I found initially confusing, is a data container and contains information on how the device communicates with the host and also holds the data packet. The Imax usb controller uses 64 byte long packets for communication. Two interrupts are used: the Endpoint Interrupt_IN, uses address 0x81; it is the information packet going from the Imax to the host (computer). Endpoint Interrupt_OUT uses address 0x01 and is information gong from host->imax.

5. USB Packet Data Capture

At this point we have all the information we need to begin to analyze the input and output information on the USB port (well, way more than we need for the goal of this project). Now we need to figure out what the two devices are transmitting to each other. The configuration information tells us only about how the two devices are communicating, not about what they are saying within the 64 byte packets of information they send back and forth, and how that information is structured in the packet.

The communication sniffer software tool, Wireshark, is one tool that can be used to get at the information within the packets. Wireshark was used extensively to monitor the USB communication traffic between the Imax and the Chargemaster 2.03 software provided by FrSky. When installing Wireshark, make sure to also install the USBPcap extension. Using Wireshark is not hard once you get through some of the tutorials on USB. Be forewarned that many of the Wireshark tutorials are way out of date with respect to loading USBPcap. If installed while installing the latest version of Wireshark, the USBPcap extension is essentially built-in. When Wireshark is started, it will examine your computer interfaces and present a list of them. Go to an interface communication by double clicking it.

In practice, the Imax was connected to a computer USB port using a standard USB to microUSB cable. Wireshark was started next.

Initially, some trial and error was necessary to discover which interface the Imax device was on. In my system, there were 4 interfaces. Open an interface and then open the Chargemaster app. Chargemaster should immediately start talking to the Imax requesting information in a rapid stream, assuming you have already connected the Imax to the host computer. If after a few moments you begin to see com traffic in the Wireshark windows, you have the right interface. If not, stop the interface by clicking the record button, close the window, and try the next interface. Once I had found the interface, it always was the one to go to.

Generally, only 15 - 60 seconds worth of communication was sufficient to either verify or obtain sufficient packet information. Chargemaster appears to send an interrupt_OUT query to the Imax ~ 0.5 seconds, so data accumulates very fast.

The real, tedious fun starts now. To determine which bytes are supplying what information requires changing charging or system conditions using Chargemaster, and recording a sequence of packets between the charger and computer. Remember that the charger not only changes charging conditions such as amperage, but is capable of handling 7 battery types, 5 different charge/discharge modes, in addition to several other functions and important system limit parameters. Thus, many boring iterations are required, even if the goal is simply to log results. For each iteration, the data must be extracted from WireShark, and conveniently displayed somewhere for comparative analysis. The problem is somewhat a combinatorial puzzle. Of course, after a small set of runs, some packets especially those going from the host to device, will be found to not change, and can be ignored, but the task still involves a significant, dedicated amount of drudge time.

With both Wireshark and Chargemaster open on the desktop, the typical iterated process was to change the conditions to be examined against previous conditions on the Chargemaster app. Pull up Wireshark, and press the shark fin icon to re-engage the USPcap interface. When the USBPcap screen appears, immediately bring up the Chargemaster screen, press the Chargemaster "Start" button, wait as data was accumulated, click the "Stop" button on the Chargemaster, and finally click the recording button on Wireshark, to stop recording packets. Using the bottom window of the USBPcap information, the "Leftover Capture Data" contained the 64 bytes in hexadecimal format. Clicking on the region, and then clicking Edit|Copy|Value each of the important host or imax packets sent was copied to an Excel worksheet and annotated with the conditions of the run.

6. Packet Analysis

The data copied to the worksheet was in the form of a single long string of hexadecimal bytes, which was inconvenient for comparison. A very simple, small Excel vba macro was written to split and expand the string on another worksheet so each byte was in a single cell; the annotations were copied as well.

From the master sheet of packets per run, the various type of packets: idle, start, settings, data, and stop, were further manually segregated to separate worksheets, for more direct comparisons. Analysis of the bytes was usually back and forth eyeball comparisons. In a couple of confusing cases. a simple ''=IF" cell formula was used to automatically compare any changed hexadecimal values between sets of data. If any of this sounds sophisticated, it was not, if you are at all familiar with Excel or Open Office spreadsheets. (Notwithstanding that the latter uses the less friendly oobasic language for calculations.)

The analysis process is inferential, that is iterative and stepwise. Data is acquired, copied, segregated, and analyzed, leading to further data gathering cycles to prove inferences or confirm byte functions.

The information provided by Milek7 was an important starter for how information was arranged in the packets received from the Imax. However, as it turned out, there are quite a few host packets that are sent that return similar or different data from the Imax, than just the data packets.

I will present all of the generalized packet structure information I found. Information on bytes that were unambiguously identified are simply stated. Any time a "?" appears in the annotations, means the byte value did not change, although it seems it is in a region where it might indicate some use, or had values changing alone or in combination with adjacent bytes, that did not result in a value that was interpretable. Therefore, there might still some work to be done, if anyone is interested.

Despite all the packet IN and OUT data shown, there is one glaring missing piece. Some settings and Imax conditions can be obtained from the Imax. However, some are available only after the Imax is started, and many others can be changed by the host using a setup packet, but a packet structure to read all the Imax current settings from one packet was not found.

7. Data Output and Input

This section will only concentrate on the communications resulting in data output either before, during, or after charging. Data during charging was the easiest to decipher because Milek7 had already defined the structure.

In all the tables, the imax-host byte values may be variable, unless specifically designated as static, for instance, the first 4 bytes from imax->host). For host->imax bytes, all byte values are static, except for the settings packet, i.e., the packet the host sends to change the Imax charger parameters.

As indicated some outputs are 2 bytes. Keep in mind these were the byte values that Chargemaster sent, which is the host app sending the information to the Imax device. Also all values except the byte number are hex values. If a single digit is given, such as 3, it must be sent as \x03 (if using imax_usb) or as 0x03.

When the Chargemaster application is started, and before the "Start" button is clicked, the following packet sequence is sent, which outputs the Imax charger system settings and the voltage of the attached battery. The software continually polls the Imax with this packet, but does not display any indication of the information it receives back.

Chargemaster started, but "Start" b button not pressed yet. "Idling". Current status of limits and attached battery.			
Byte	**host->imax**	**imax->host**	**Use, imax->host**
0	0f	0f	Request type; static value

1	3	25	Request type; static value
2	5a	5a	Request type; static value
3	0	0	Request type; static value
4	5a	3	cycle delay, min
5	ff	0	?
6	ff	01 90	max chrg time
7	0		
8	0	0	?
9	0	07 d0	max mV
10	0		
11	0	0	key buzzer; 0 = off; 1 = on
12	0	0	sys. Buzzer; o= off; 1 = on
13	0	2a f8	input volt, low mv (indicates power into Imax
14	0		
15	0	0	?
16	0	0	?
17	0	3c	max. T?
18	0	24	Current Battery Voltage, mV
19	0	2c	
20	0	2	?
21	0	55	?
22	0	1	?
23	0	28	?
24	0	1	?
25	0	19	?
26	0	1	?
27	0	18	?
28	0	1	?
29	0	23	?
30	0	1	?
31	0	2d	?
32	0	0	?
33	0	0	?
34	0	0	?
35	0	0	?
36	0	0	?
37	0	0	?
38	0	78	?
39	0	ff	?
40	0	ff	?
41-63	0	0	unused

Another packet was sent from the host to Imax when the Chargemaster "Start" button was pressed. Below is the output observed over different runs.

Start? or Wait? Packet.

Byte	host -> imax	imax -> host	Notes
0	0f	0f	host: always sent; Imax: always returned
1	3	0f	host: always sent; Imax: always returned
2	5f	5f	host: always sent; Imax: always returned
3	0	0	host: always sent; Imax: always returned
4	5f	2	host: always sent; Imax: always returned
5	ff	0	host: always sent; Imax: always returned
6	ff	0	host: always sent; Imax: always returned
7	0	0	host: always sent; Imax: always returned
8	0	3c	host: always sent; Imax: always returned
9	0	14	host: always sent; Imax: always returned
10	0	1	host: always sent; Imax: always returned
11	0	2	host: always sent; Imax: always returned
12	0	0e	host: always sent; Imax: always returned
13	0	7	host: always sent; Imax: always returned
14	0	ff	host: always sent; Imax: always returned
15	0	ff	host: always sent; Imax: always returned
16	0	c7	host: always sent; Imax: always returned
17	0	ff	host: always sent; Imax: always returned
18	0	ff	host: always sent; Imax: always returned
19-63	0	variable	host: always sent; Imax: varies, noncorrelated

As indicated, the first 18 bytes returned the same values in either direction. The remaining bytes returned values that could not be correlated with any of the settings applied to the Imax.

Data output is triggered by the following host->imax packet structure. The Imax returns a packet with the data structured as indicated:

Data accumulation. Host Interrput_OUT packet and Imax Interrupt_IN packet hexadecimal byte values.

Byte	host -> imax	imax -> host. example	Use, imax-host
0	0f	0f	Request type; static value
1	3	22	Request type; static value
2	55	55	Request type; static value
3	0	0	Request type; static value
4	55	1	Run State 1 = running; 2 = user stop; 3 = imax finish
5	ff	00 06	Energy
6	ff		
7	0	00 01	Timer (seconds)
8	0		
9	0	25 53	Voltage; mV
10	0		
11	0	00 00	Current, mA
12	0		
13	0	00	ext. T?
14	0	19	int. T deg C, but may be affected by C or F setting

15	0	00 00	error notice, 0 = ok; error = ffc7; dec.: 65479
16	0		
17	0	ff ff	Cell 1; LIPO, LiFe, LiIO, LiHv only
18	0		
19	0	60 02	Cell 2; LIPO, LiFe, LiIO, LiHv only
20	0		
21	0	4e 01	Cell 3; LIPO, LiFe, LiIO, LiHv only
22	0		
23	0	28 01	cell 4; LIPO, LiFe, LiIO, LiHv only
24	0		
25	0	1a 01	cell 5; LIPO, LiFe, LiIO, LiHv only
26	0		
27	0	18 01	cell 6; LIPO, LiFe, LiIO, LiHv only
28	0		
29	0	23 01	cell 7?
30	0		
31	0	2e	cell 8?
32	0	0	?
33	0	1	?
34	0	0	?
35	0	47	?
36	0	ff	?
37	0	ff	?
38	0	9a	?
39	0	ff	?
40	0	ff	?
41-63	0	0	Unused

As stated in the table, only those bytes which have been verified to have some use or may be used are shown. Packets 41-63 are not shown to conserve space; they were always 0x00 in both host and Imax packets. The host->imax byte values do not change, nor do the first four imax->host bytes. Sending the host->imax packet results in the format shown for imax->host output. Several of the cells have 2 byte values. Whether what appears as Cell 7 and Cell 8 really do something is unknown. Certainly the balance ports only have pin outs for 6 cells.

The Chargemaster sends several data OUT requests in rapid succession, after the settings packet (see below) was sent. Wireshark showed anywhere between 2-4 host packets sent before the Imax sent an IN data packet, with usually 3 as the most common number observed. As already mentioned, it appeared the host sent a request packet ~0.5 second intervals. Of course, the byte values in the imax->host for the battery data will be changing; the values shown are only examples.

The last data sequence that was found was when the chargemaster stop button is pressed. A single host->imax packet is sent, and a single imax packet is returned as follows:

Chargemaster "Stop" button pressed. Single packet sent and single received.			
Byte	**host->imax;**	**imax->host (example values)**	**Use, imax->host**

0	0f	f0	Request type; static value
1	3	ff	Request type; static value
2	fe	ff	Request type; static value
3	0	0	Request type; static value
4	fe	1	? 1 = NiCd, NiMH; 4 = LiPO, LiIO, LiHV
5	ff	00 02	Capacity input, mah
6	ff		
7	0	00 10	timer, seconds
8	0		
9	0	15 04	voltage, mV
10	0		
11	0	02 53	current mA?
12	0		
13	0	0	
14	0	18	int temp
15	0	0	
16	0	0	
17	0	ff	Cell 1, mV
18	0	ff	
19	0	60	Cell 2, mV
20	0	2	
21	0	43	Cell 3, mV
22	0	1	
23	0	28	cell 4, mV
24	0	1	
25	0	19	cell 5, mV
26	0	1	
27	0	18	cell 6, mV
28	0	1	
29	0	23	cell 7?
30	0	1	
31	0	2d	cell 8?
32	0	0	?
33	0	1	?
34	0	0	?
35	0	40	?
36	0	ff	?
37	0	ff	?
38	0	ab	?
39	0	ff	?
40	0	ff	?
41-63	0	0	unused

After the stop packet is sent and the Imax packet received, Chargemaster sends the same idling mode requests, as in the second table above.

With the information in the three tables above, developing a python based logging package was relatively straightforward.

Application of these host packets is 'safe'; they do not directly change settings of the imax, only explore the value retained by the Imax. The logging program versions use these packets.

8. Notes on full Imax B6 mini control software

The downside of expanding the utility of the applications to Imax control is the even more dedicated analysis time to examine the settings control packets from the host to Imax. The table below represents the resolved structure of packets to set or control conditions of the Imax B6 mini charger.

Packet Structure to Change Imax settings using host.

Byte	host->imax (example)	Description. (All values in bytes)
0	0f	Direction host_imax; always 0x0f
1	16	Operation Type? Set imax conditions; always 0x16
2	5	? Always 0x05?
3	0	? Always 0?
4	4	Battery Type: LIPO = 0x00; LiIO = 0x01; LIFe = 0x02; LiHV = 0x03; NIMH = 0x04; NiCd=0x05; Pb = 0x06
5	8	Number of Cells
6	0	Charge/Discharge Modes: LiPO, LIHv, LiIO, LiFe: CHRG=0x0 0; DISCHRG=0x01; STORAGE=0x0 2; FAST CHRG=0x0 3; BAL. CHRG=0x04. NiMH, NiCd: CHRG=0x0 0; AUTO CHRG=0x01; DISCHRG=0x0 2; REPEAK= 0x03; CYCLE=0x04.
7	02 58	Charging Current; milliamps: 2 bytes
8		
9	00 c8	Discharging Current; milliamps: 2 bytes
10		
11	04 4c	Minimal Voltage/cell; millivolts: 2 bytes
12		
13	00 04	Default sensitivity; either peak sensitivity (NiMH/NICd) = 0x0004 or for LIXX = max. mV: LIPO = 0x1068, LiIO = 0x1004, LiFe = 0x0e74, LiHV = 0x10fe, Pb =0x00
14		
15	0	cycle set C/D =0x0 0; D/C = 0x01
16	0	Number of cycles (max. value =0x05); set to value ONLY IF BYTE 4 SET TO 0x04(NiMH) OR 0x05(NiCd) AND BYTE 6 SET TO 0x04(CYCLE)
17	0	unused ? No change observed
18	0	unused ? No change observed
19	0	unused ? No change observed
20	0	unused ? No change observed
21	0	unused ? No change observed
22	0	unused ? No change observed
23	87	Byte check sum. Calculated as: hexadecimal value of SUM(byte2:byte22); WHILE SUM>256: SUM =(SUM-256); else: SUM.(See code for a streamlined method to get low order byte.)
24	ff	unused ? No change observed
25	ff	unused ? No change observed
26-64	0	unused ? No change observed
		return packet is same as hoist-imax settings packet

There are a couple of caveats in this data: not every possible combination was explored, because some battery types were not available. A question mark in the description may imply several things, depending on any other included descriptive information. In most cases, the value given was never observed to change within the scope of the many trials to determine what bytes controlled what settings.

Byte 23 is a check sum, calculated in a special way as indicated in the description. All the data collected was reconciled to the calculation method indicated. How it is used internally is unknown. Better to calculate it correctly and put it in, then ignore it. Quite a few transmits of the setting packet were necessary during development. In no case, did the Imax fail, nor fry itself or a battery. Nonetheless, your are warned about using at your own risk.

Extending the simple logging application to Imax control required a lot more effort. The above tables provide the basic run information for defining the run control conditions and starting the Imax. That is the simple stuff. Beneath the control packet information comes the responsibility of making sure the conditions are set up properly to not cause the charger to burn itself up, or cause a battery fire. To a large extent, the Imax apparently controls these limits. However, all the under-the-hood controls on conditions are not clear. For example, the factory limit setting for maximum capacity is very high, while the time limit setting is fairly short, in sort of a ying-yang relationship. Not properly setting these limits could lead to dangerous conditions. A chart in the Imax manual shows the various limits for battery types that are built into the charger. To be safe, these values should also be calculated and checked for any input conditions that a user wishes to inputs. For instance, the battery type, the maximum voltage/cell, the maximum mah, and the number of cells are all related, and some sort of master table of these parameters (such as a built-in dictionary list, or as a separate file) would be advisable to calculate operational limits as a further check on safe conditions, before sending a set of run conditions to the charger. Calculating the conditions for each case is not difficult, but does require tedious attention to detail, especially regarding the relationship between battery type, maximum and minimum voltage per cell, number of cells, maximum capacity of the battery pack. maximum or at least optimal C rating, maximum Temperature, maximum time, etc.

The packet generated with the above format does not change the limit conditions: maximum charge time, charge delay time, maximum capacity allowed, or Imax buzzers. As already provided in the table for the "idling" request packet from the host, the limit values can be easily read. Writing the limit conditions to the Imax is more complicated. A sequence of four packets is always sent from the host to the Imax charger to update the limits and buzzer parameters. The host to Imax packet sequences are displayed below:

Set Maximum Capacity limit (mah)		
Byte	**host->imax**	
0	**0x0f**	Always for Saving limits
1	**0x07**	Seems to represent low order byte position for max. mah
2	**0x11**	Always
3	**0x02**	Set to change max. mah
4	**0x00**	Always
5	**0x00**	Use as limit: 0x00 is off; 0x01 is on (example do not use as limit)
6	**0x07**	high order byte value for max mah
7	**0xd0**	with byte 6 = max mah; example shown: 2000 mah
8	**0xeb**	sum of byte2 to byte 7
9	**0xff**	always returns
10	**0xff**	always returns
11-64	**0x00**	Always

Set Maximum Charge Time		
Byte	host->imax	
0	0x0f	Always for Saving limits
1	0x07	represents low order byte position
2	0x11	Always
3	0x01	max time
4	0x00	Always
5	0x00	Use as limit: 0x00 is off; 0x01 is on (example: do not use as limit)
6	0x00	high order byte for max. time
7	0xb4	with byte 6 = max. time in minutes: example is 180 min
8	0x21	sum of bytes2 to byte7
9	0xff	always returns
10	0xff	always returns
11-64	0x00	always

Set Discharge/Charge_Delay		
Byte	host->imax	
0	0x0f	Always for Saving limits
1	0x05	appears to indicate low order byte position value to be changed
2	0x11	Always
3	0x00	for DC_Delay
4	0x00	Always
5	0x1a	D/C delay in minutes, example = 10 min.
6	0x1b	sum byte2 to byte 5; example 0x1b
7	0xff	always
8	0xff	always
9-64	0x00	always

Set Buzzers		
Byte	host->imax	
0	0x0f	Always for Saving limits
1	0x06	represents low order byte position of last value, e.g., Sys. Buzzer.
2	0x11	Always
3	0x03	for buzzers
4	0x00	Always
5	0x01	key buzzer 0x00 = off, 0x01 = on.
6	0x01	sys. buzzer is 0x00 = off; 0x01 =on
7	0x16	sum of byte2 to byte6; (note shift in sum)
8	0xff	always returns
9	0xff	always returns
10-64	0x00	always

These limit changing packets were not incorporated into the run_*imax.py*. The limits would have required the addition of two more dropdown and four radio button widgets. Just how useful this would be was not clear, at the expense of adding to an already expanded plethora of settings to manage.

Along with the control function, software control of the Imax charger offers the ability to create an extensive library of settings for various configurations. The Imax has a built-in storage memory, and the Chargemaster software extends the number of possible combinations, through some sort of storage. (I found the Chargemaster memory system confusing.)

The method chosen here for storage and retrieval was the object relational mapper, SQLalchemy, with an sqlite database.

The partial use of the ini file to store only some initial settings, and other settings are managed in *run_imax.py* was based on personal preference. There is no special reason for this arrangement. There is an ini generator file that can also be used to rebuild the imaxconfig.ini file, *make_imaxconfig_file.py.*

The annoyance of not being able to re-choose an already chosen program comes about because the select widget only allows an on_change event. I could not figure out a way to return the originally chosen program settings, without first choosing a different set of program settings.

Plotting can be improved, so that it would start with 1

The code likely has several areas that could be improved to reduce redundancy, and tighten up the code to run more efficiently. It is at the stage, *"it all seems to work' and I am done fooling with it".*