**Battery Cycler Communications Protocol 0.1**

The order that data is transmitted and received is shown in listed in the order that they are received / transmitted.

Communications from the host PC to the battery cycler system and from the battery cycler system to the host PC are very similar. They use the same start byte as shown in , and both end with a 16bit CRC. The data contained in the message is of a fixed length and specific to the type of transmission as detailed below.

Table : Byte order

|  |
| --- |
| Start Byte |
| X number of data bytes |
|
| CRC upper byte |
| CRC lower byte |

Table : Start Byte

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit # | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Contents | *Unused* | | | CH# | Type of transmission | | | |

**Types of transmissions (HOST PC)(0-7):**

1. **Basic request:** A basic request is an action request from the host PC that may or may not be completed while the system is busy. The basic request transmission contains one data byte that contains the request number. Requests 0-127 may be processed while the system is busy. Requests 128-255 may not be processed while the system is busy. See for a detailed list of basic requests.
2. **Variable update:** A variable update message has 3 data bytes; the first contains the variable ID that is to be updated. Followed by the 16bit integer.
3. **Profile request:** A profile request message has only 1 data byte; it contains the profile ID that is to be started. A profile request can only be made while the system is idle.

**Types of transmissions (SYSTEM)(8-15):**

1. **System Idle:** When a channel is Idle its heartbeat transmission will have 4 data bytes containing two temperatures. Cell temp and chamber temp. In units of 0.1 deg C.
2. **System Active:** When a channel is active its heartbeat transmission will have a large number of bytes of data, this data is outlined in table 2.
3. **System Fault:** When a channel is in the fault state its heartbeat transmission will have 1 data byte that contains the specific fault code the channel is experiencing.

Table : Data provided in active channel heartbeat

****

Table : List of Basic Requests

|  |  |  |
| --- | --- | --- |
| **Request #** | **Request Name** | **Description** |
| **0** | **Stop** | **Halts testing on the specified channel, sets output to 0. Status becomes idle.** |
| **1** | **Start** | **Will enable the specified channel at whatever step and profile it was stopped at. (ex. Resume from a fault)** |
| **2** | **Reset energy counters** | **Zeros out the Charge, load, and SOC, counts for the specified channel.** |
| **3** |  |  |
| **4** |  |  |
| **5** | **Test mode** | **Outputs ramp data for channels 0 and 1 until stopped** |
|  |  |  |

Table : List of updatable variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable #** | **Variable Name** | **Units / type (Range)** | **Description** |
| **0** | **Chamber Temp** | **0.1 Deg C / int (+- 3,200)** | **Setpoint for chamber** |
| **1** | **Max cell temperature** | **0.1 Deg C / int (+- 3,200)** | **Fault temp for cell** |
| **2** | **BSF (battery scale factor)** | **- / Unsigned int (64,000)** | **BSF for relative profiles** |
| **3** | **Max cell voltage** | **mV / unsigned int (64V)** | **Battery voltage for high voltage fault** |
| **4** | **Min cell voltage** | **mV / unsigned int (64V)** | **Battery voltage for low voltage fault** |
| **5** | **PID kP** | **TBD** |  |
| **6** | **PID kI** | **TBD** |  |
| **7** | **PID kD** | **TBD** |  |
|  |  |  |  |

Table : List of Profile Requests

|  |  |  |
| --- | --- | --- |
| **Request #** | **Profile Name** | **Description** |
| **0** |  |  |
| **1** |  |  |
| **2** |  |  |
| **3** |  |  |
| **4** |  |  |
| **5** |  |  |
| **6** |  |  |

Table : List of System Faults

|  |  |  |
| --- | --- | --- |
| **Fault #** | **Fault Name** | **Description** |
| **0** |  |  |
| **1** |  |  |
| **2** |  |  |
| **3** |  |  |
| **4** |  |  |
| **5** |  |  |
| **6** |  |  |

**Schedule and Program files:**

**Definition:**

Let’s start with the definition of each term relative to the battery cyclers operation.

A program is a sequential list of commands that is what the scheduler is going to read and transmit to the battery cycler system to be processed. It is processed one line at a time and the next line cannot be run until the previous one has been finished. A battery researcher can use a program to tell the system what tests to run and in what order, when to save a new data file and what to call it, etc. You can also have as many different programs and you desire.

A schedule is very similar to a program. A schedule is simply a list of programs and how many times to run each one. Again this is a list where one program must be completed before the next one can begin.

Why do we need both a schedule and a program if they are almost the same thing? This is because, as the name would imply cycling batteries requires a lot of cycling, or a lot of repetitive actions. So by having a separate schedule file where we can specify that we want one program to be repeated 400 times before we move on we just saved your fingers a lot of copy and pasting and it becomes much easier to modify the length of a test.

**Formatting:**

So how are these schedules and programs formatted? They are both comma delaminated files that are formatted as shown in the examples below. A program being the lower level file list the exact transmission type, the ID of the request #, variable#, or profile# to be run. As well as any additional data necessary; a variable value for updating variables, and a temperature for running profiles.

There is one very special line that can be used in a program file that does not actually send a transmission to the arduino. These commands control the scheduler program on the host pc directly. Currently the only command that does this is transmission type number 16. This specifies to save all the previously collected data that is currently being buffered if ram. It is followed by a description that the file name will end with. The file naming structure is as follows:

<Schedule name>\_<program name>\_<program cycle#>\_<description>

//The program file is written in a comma delaminated format as follows:

// <Transmission Type>, <Request# / Variable# / Profile #>, <Additional data if needed>

//Example Program File

1, 1, 800 // setting max temp

1, 2, 650 // setting the BSF

1, 3, 4300 // setting upper vlim

1, 4, 2200 // setting lower vlim

0, 2 // clearing energy counter

2, 5, 500 // run profile number 5 at a temp of 50 deg C

16, HPPC\_Charge // Setting the file name for following data collected ending with “HPPC\_Charge”

0, 2

2, 13, 500

2, 13, 500

2, 13, 500

2, 13, 500

16, HPPC\_Cycle

The higher level schedule file only lists two things; a programs name, and the number of times to run it.

Note: When a schedule file refers to a program file it should be a relative directory. IE assume the program file is in the same directory as the schedule file so just list its name.

//The schedule file is written in a comma delaminated format as follows:

// <Program Name>, <# of cycles>

//Example Schedule File

Table\_5\_CD, 60

HPCC, 1

HF\_IR, 1

Table\_5\_CD, 60

HPCC, 1

HF\_IR, 1

Table\_5\_CD, 60

HPCC, 1

HF\_IR, 1

**Sending transmissions to the Arduino and error checking:**

Using CRC to check data from host PC and acknowledging receipt of good data:

below shows the flow diagram for sending transmissions to the Arduino. We begin by checking if there is a fault, if there is a fault we do not want to send any more information to the Arduino until the fault is cleared.

If there is not a fault we need to check of the channel we are communicating with is idle or not.

If the channel is idle we can always assume it is ok to send the next line. If the channel is active / busy then we must first check if the next line to be sent is one that can be sent while the system is active / busy. Those commands are basic requests with an ID of 0-127 as outlined earlier.

Once we have been able to send a transmission we do not send another transmission until we receive a ACK flag from the arduino. The ACK flag is set on the start byte of the transmission from the arduino when it has read the transmission and the CRC has checked OK.

If we do not receive a ACK within 5seconds ( 5 heartbeats ) then consider the transmission lost or corrupted and attempt sending the same thing again.

Once we have received an ACK for the transmission the process is finished and we can begin waiting to transmit the next line.



Figure

**Notes about system operation**

Chamber temp setting: When a request for changing a chamber temp is sent in a message to CH-0 or CH-1 it is referring to the same setting as there is only one chamber!

IF one channel requests a temperature change that is not the current set temp, and the other channel is busy. It will wait until the other channel has completed its current cycle and then pause it. Change the temperature and run the original requests profile.

All requests besides profiles can be run while the channel is being held for the different temperature.