CIWS-WM-Node

Software Documentation

Software ver. 1.0.0

Software Files

The CIWS-WM-Node software is comprised of the following files:

- `cmdline.txt`
- `config.txt`
- `LoggerAutoRun.py`
- `LoggerReadRom.py`
- `LoggerReportSwap.py`
- `LoggerShell_CLI.py`
- `logger.c`
- `rc.local`
- `setup.py`
- `setup.sh`

The file `setup.sh` is used to build and setup the datalogger software by doing the following:

- Prompt user to configure WiFi
- Install dependencies
- Copy configuration files to root directories
- Build the python module Logger defined in `logger.c`
- Power off the Raspberry Pi

The WiFi configuration is important for a few reasons:

1. Dependencies are installed from the internet
2. The device is accessed via SSH
3. Internet connectivity is planned for the near future

The dependencies are:

- `python-dev`
- `wiringpi`

The following files are copied into root directories, and are used to configure Raspbian on start-up:

- `cmdline.txt (copied to /boot/)`
- `config.txt (copied to /boot/)`
- `rc.local (copied to /etc/)`

The python (`.py`) files are the interface to the functions in `logger.c`. The user interacts with these using the `LoggerShell_CLI.py` python script, which is run every time a user logs in.

GPIO Functions: `logger.c`

The code in `logger.c` comprises the Logger module used in the other python scripts. There are 17 functions in this file:

- `init`
- `initPins`
- `bufferMax`
- `setRomBusy`
- `setPowerGood`
- `loadData`
- `reportSwap`
- `setRomFree`
- `setPowerOff`
- `writeToFile`
- `storePeriod`
- `setID`
- `setSiteNumber`
- `getID`
- `getSiteNumber`
- `setMeterResolution`
- `getMeterResolution`

These functions mainly deal with GPIO and file I/O operations using the following libraries:

- `wiringPi.h`
- `wiringPiSPI.h`
- `wiringSerial.h`

The following sections describe the individual functions in this file.

Initializing the Module: `init()`

The function `init()` initializes the wiringPi GPIO library, the wiringPi SPI library, and the wiringPi Serial library. The SPI module is initialized to communicate at 2 MHz, and the Serial module is initialized to communicate at 9600 Baud. This function must be called at the beginning of any script utilizing the `Logger` module.

Initializing the GPIO pins: `initPins()`

This function sets up the initial values of the GPIO pins. There are two GPIO pins this function initializes:

- `ROM_BUSY` (Broadcom GPIO 24)
- `POWER_GOOD` (Broadcom GPIO 25)

`ROM_BUSY` and `POWER_GOOD` are both initialized as logic low (0 V).

Read Maximum Buffer Size: `bufferMax()`

This function simply returns the variable BUFFER_MAX. This value denotes the maximum amount of data the Raspberry Pi will read from the EEPROM chip; however, the actual amount read depends on the value given in the EEPROM chip, as is demonstrated later.

Request EEPROM: `setRomBusy()`

This function sets the `ROM_BUSY` pin `HIGH`, which tells the microcontroller that the Raspberry Pi is reading the EEPROM. The microcontroller yields the EEPROM chip ONLY IN THE FOLLOWING SITUATIONS:

1. The Raspberry Pi was just activated
2. The Raspberry Pi instructed the microcontroller to stop a logging session

The microcontroller will not honor this request in any other situations.

Report Power On: `setPowerGood()`

This function sets the `POWER_GOOD` pin `HIGH`, which tells the microcontroller that the Raspberry Pi has successfully powered on.

Read Data from EEPROM: `loadData()`

This function reads data from the EEPROM chip. The function allocates an array to hold the EEPROM data, then calls a SPI transaction, which transfers data from the EEPROM chip to the data array. The following table shows the layout of data in the EEPROM:

Byte	Field
0	Number of records (MSB)
1	Number of records
2	Number of records (LSB)
3	Starting timestamp (year)
4	Starting timestamp (month)
5	Starting timestamp (day)
6	Starting timestamp (hour)
7	Starting timestamp (minute)
8	Starting timestamp (second)
9	Data byte 0
10	Data byte 1
11	Data byte 2
...	...
N	Data byte N - 9

Each data byte is the number of pulses during a configurable time interval.

Swap Report with Microcontroller: `reportSwap()`

This function swaps “report” data with the microcontroller. This data consists of the following:

Byte	Field	R/W
0	Year	R/W
1	Month	R/W
2	Day	R/W
3	Hour	R/W
4	Minute	R/W
5	Second	R/W
6	Pulses in previous sample	R
7	Total pulses (MSB)	R
8	Total pulses	R
9	Total pulses (LSB)	R
10	Logging flag	R/W
11	Time interval	R/W

The data swap is done over a serial UART connection. The data messages are interlaced; in other words, the Raspberry Pi sends one byte, and the microcontroller sends the corresponding byte. The Raspberry Pi can overwrite data fields marked R/W by sending any data value other than `0xFF (READ)` or `0xEE (START)`. Each data swap is initiated by the Raspberry Pi by sending a START byte to the microcontroller. Sending a new START byte will always initiate a new swap, even if one is in-progress. Any data written during the unfinished swap will still be written, while all other fields will remain untouched.

Release EEPROM: `setRomFree()`

This function simply sets the `ROM_BUSY` pin `LOW`, signaling to the microcontroller that the Raspberry Pi has finished reading the EEPROM.

Report Power Off: `setPowerOff()`

This function sets the `POWER_GOOD` pin `LOW`, signaling to the microcontroller that the Raspberry Pi is shutting down. The microcontroller will cut power to the Raspberry Pi roughly 12 seconds after receiving this information.

Write Data to File: `writeToFile()`

This function is responsible for writing data to a file on the Raspberry Pi. The function parses the data for information and writes it all to the file. This file is similar in format to the files in the CIWS-MWM-Logger project.

Time Interval: `storePeriod()`

This function is used to store the time interval set in the microcontroller. The time interval is the time (in seconds) between data records in the EEPROM. This value is stored on the Raspberry Pi in a file called `clockPeriod.txt`, as well as on the microcontroller. The microcontroller reports the interval during a report swap.

Datalogger ID: `setID()` and `getID()`

These functions are used for setting and reading the device's ID number. The ID number is stored on the Raspberry Pi in a file called `IDconfig.txt`.

Datalogger Site: `setSiteNumber()` and `getSiteNumber()`

These functions are used for setting and reading the device's site number. The site number is stored on the Raspberry Pi in a file called `siteConfig.txt`.

Meter Factor: `setMeterResolution()` and `getMeterResolution()`

These functions are used for setting and reading the device's meter resolution. The meter resolution is a conversion factor between the number of pulses and the volume of water through the meter. This value is stored on the Raspberry Pi in a file called `meterConfig.txt`.

Python Scripts

The software contains several python scripts which may be customized for different uses; however, the default scripts should suffice for most applications.

Autonomous Functionality: `LoggerAutoRun.py`

The microcontroller powers the Raspberry Pi every morning at 12:00 AM. The Raspberry Pi must then autonomously read, process, and store the EEPROM data with no intervention from the user. The Raspberry Pi must also power itself back off again. All of this is done by the script `LoggerAutoRun.py`. This script is run every time the Raspberry Pi is powered on by placing a call to this script in the file `/etc/rc.local`. This script handles the power and EEPROM signals between the Raspberry Pi and the microcontroller, reads EEPROM data, and stores it in a file. If the power-on event occurred at 12:00 AM, the script assumes the power-on was automatic, and the Raspberry Pi is shut down; otherwise, it is assumed the user powered on the Raspberry Pi and the Raspberry Pi is not shut down.

Reading the EEPROM: `LoggerReadRom.py`

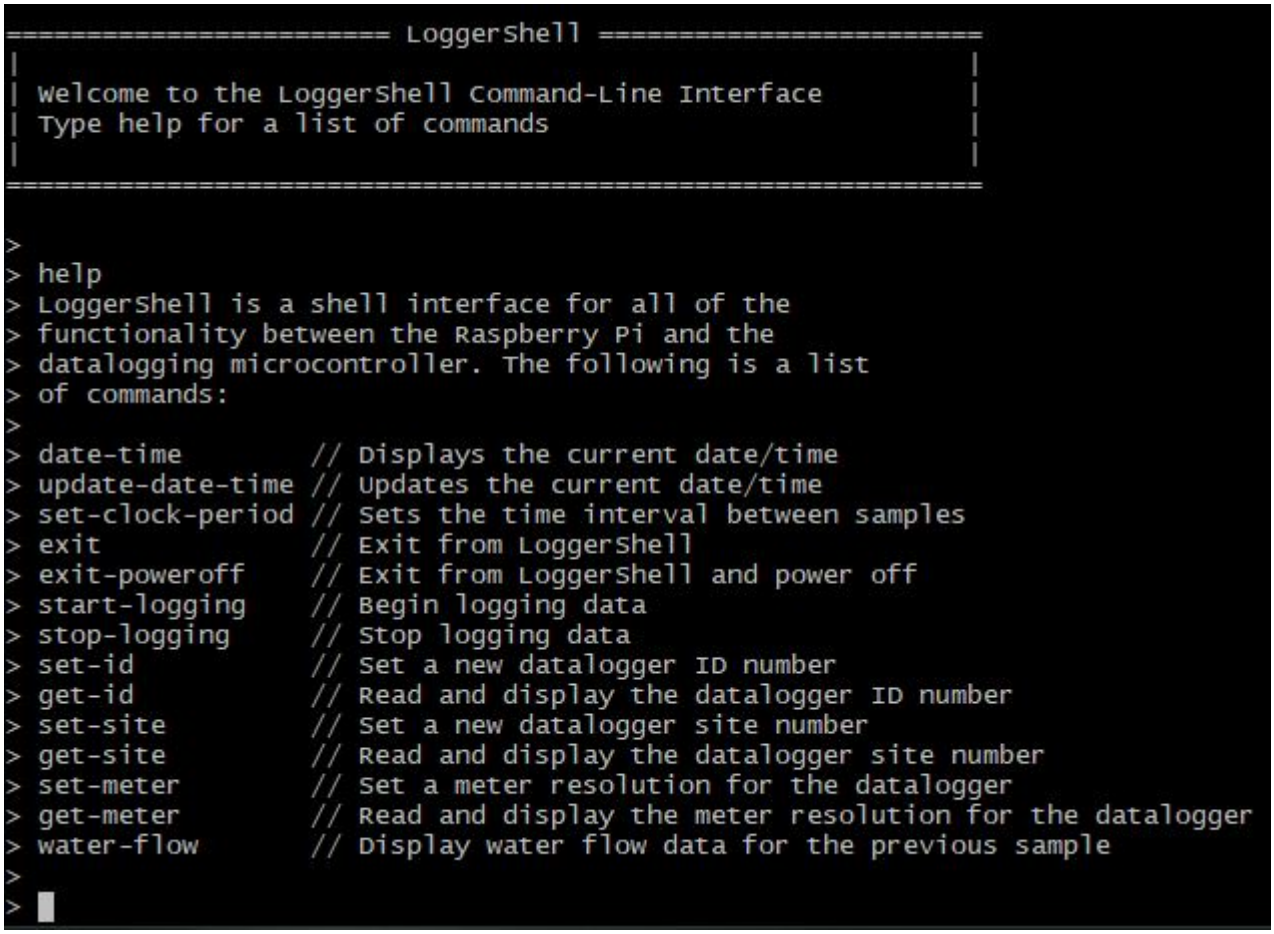
The script `LoggerReadRom.py` is very similar to `LoggerAutoRun.py`. The difference is that `LoggerAutoRun.py` is only run on power-up, while `LoggerReadRom.py` is used to read the EEPROM when a logging session is ended, and is called by LoggerShell_CLI.py.

Swapping Reports: `LoggerReportSwap.py`

The script `LoggerReportSwap.py` is used to swap reports with the microcontroller by calling `Logger.reportSwap()` (see details in previous sections). This script is generally called by `LoggerShell_CLI.py` when a user command requires information to be read from the microcontroller.

User Interface: `LoggerShell_CLI.py`

`LoggerShell_CLI.py` is the interface between the user and all of the functionality of the WM-Node datalogger. This python script calls the other python scripts described, with the exception of LoggerAutoRun.py, which is called by rc.local on power-up. The interface is shown in the following screenshot of LoggerShell_CLI.py displaying the help menu:



The datalogger ID, site #, and meter resolution are all set and read by running `set-id` and `get-id`, `set-site` and `get-site`, and `set-meter` and `get-meter`. Date and time can be viewed by running `date-time`, or updated by running `update-date-time`. The command `set-clock-period` sets the time interval between data records. A logging session is started by running `start-logging`, or stopped by running `stop-logging`. During a logging session, `water-flow` can be called to view water flow data. Finally, `exit` and `exit-poweroff` end the `LoggerShell_CLI.py` script. Just running `exit` allows the user access to the Linux file system, while `exit-poweroff` will shut down the datalogger.