Here's an example app that logs software polled data to a csv file. Using the MCC 118, the data is collected using a_in_scan() in background continuous modes and demonstrates writing data in Python in blocks.
The file name is auto generated based on date and time for example: yyyy_Month_dd_hhmmss.csv
The data is stored in $ /home/pi/Documents/Measurement_Computing/log_files

---

```
#!/usr/bin/env python
#  -*- coding: utf-8 -*-

from __future__ import print_function
from sys import stdout
from time import sleep
from daqhats import mcc118, OptionFlags, HatIDs, HatError
from daqhats_utils import select_hat_device, enum_mask_to_string, \
    chan_list_to_mask
from datetime import datetime, timedelta

import os
import csv    #this is the import to make csv file creation simple.
import errno

READ_ALL_AVAILABLE = -1

CURSOR_BACK_2 = '\x1b[2D'
ERASE_TO_END_OF_LINE = '\x1b[0K'

def main():
    """
    This function is executed automatically when the module is run
directly.
    """

    # Store the channels in a list and convert the list to a channel mask
that
    # can be passed as a parameter to the MCC 118 functions.
    channels = [0, 1, 2, 3]
    channel_mask = chan_list_to_mask(channels)
    num_channels = len(channels)
    samples_per_channel = 0
```

```
    options = OptionFlags.CONTINUOUS
    scan_rate = 1000.0


    try:
        # Select an MCC 118 HAT device to use.
        address = select_hat_device(HatIDs.MCC_118)
        hat = mcc118(address)

        print('\nSelected MCC 118 HAT device at address', address)

        actual_scan_rate = hat.a_in_scan_actual_rate(num_channels,
scan_rate)

        print('\nMCC 118 continuous scan example')
        print('    Functions demonstrated:')
        print('         mcc118.a_in_scan_start')
        print('         mcc118.a_in_scan_read')
        print('         mcc118.a_in_scan_stop')
        print('    Channels: ', end='')
        print(', '.join([str(chan) for chan in channels]))
        print('    Requested scan rate: ', scan_rate)
        print('    Actual scan rate: ', actual_scan_rate)
        print('    Options: ', enum_mask_to_string(OptionFlags, options))

        try:
            input('\nPress ENTER to continue ...')
        except (NameError, SyntaxError):
            pass

        # Configure and start the scan.
        # Since the continuous option is being used, the
samples_per_channel
        # parameter is ignored if the value is less than the default
internal
        # buffer size (10000 * num_channels in this case). If a larger
internal
        # buffer size is desired, set the value of this parameter
accordingly.
        hat.a_in_scan_start(channel_mask, samples_per_channel,
scan_rate,
                            options)
```

```
        print('Starting scan ... Press Ctrl-C to stop\n')

        # Display the header row for the data table.
        print('Samples Read    Scan Count', end='')
        for chan, item in enumerate(channels):
            print('    Channel ', item, sep='', end='')
        print('')

        try:
            read_and_display_data(hat, num_channels)

        except KeyboardInterrupt:
            # Clear the '^C' from the display.
            print(CURSOR_BACK_2, ERASE_TO_END_OF_LINE, '\n')
    except (HatError, ValueError) as err:
        print('\n', err)

def read_and_display_data(hat, num_channels):
    """
    Reads data from the specified channels on the specified DAQ HAT
devices,
    updates the data on the terminal display and writes the data to a
.csv
    file.  The reads are executed in a loop that continues until the user
    stops the scan or an overrun error is detected.

    Args:
        hat (mcc118): The mcc118 HAT device object.
        num_channels (int): The number of channels to display.

    Returns:
        None

    """
    total_samples_read = 0
    read_request_size = READ_ALL_AVAILABLE

    basepath = '/home/pi/Documents/Measurement_Computing'
    mypath = basepath + '/Scanning_log_files'

    # When doing a continuous scan, the timeout value will be ignored
in the
    # call to a_in_scan_read because we will be requesting that all
```

```
available
    # samples (up to the default buffer size) be returned.
    timeout = 5.0

    # Read all of the available samples (up to the size of the
read_buffer which
    # is specified by the user_buffer_size).  Since the
read_request_size is set
    # to -1 (READ_ALL_AVAILABLE), this function returns immediately
with
    # whatever samples are available (up to user_buffer_size) and the
timeout
    # parameter is ignored.

#============================================================


    # If the scan starts, create a file name based upon current date and
time.
    # Generate the full path
    # to where to write the collected data as a .csv file.  Open the file
    # begin writing the data to the file.  When done, close the file.

    try:
if os.path.exists(basepath):
if not (os.path.exists(mypath)):
os.mkdir(mypath)
else:
os.mkdir(basepath)
os.chdir(basepath)
os.mkdir(mypath)
    except OSError as exc:
raise

    os.chdir(mypath)
    fileDateTime = datetime.strftime(datetime.now(),
"%Y_%B_%d_%H%M%S")
    fileDateTime = mypath + "/" + fileDateTime + ".csv"
    csvfile = open(fileDateTime, "w+")
    csvwriter = csv.writer(csvfile)
```

```
    while True:
        read_result = hat.a_in_scan_read(read_request_size, timeout)

        # Check for an overrun error
        if read_result.hardware_overrun:
            print('\n\nHardware overrun\n')
            break
        elif read_result.buffer_overrun:
            print('\n\nBuffer overrun\n')
            break

        samples_read_per_channel = int(len(read_result.data) /
num_channels)
        total_samples_read += samples_read_per_channel

        totalSamples = len(read_result.data)
        print("\r MyTotalSamples = %d\n" % totalSamples)

        # Display the last sample for each channel.
        print('\r{:12}'.format(samples_read_per_channel),
              ' {:12} '.format(total_samples_read), end='')

        if samples_read_per_channel > 0:
            index = samples_read_per_channel * num_channels -
num_channels
            print("\r Index = %d\n" % index)

            new_index = 0
            myArray=[] #create an empty array
            for i in range(0, totalSamples, num_channels):
myArray.append([])  #add a row to the array (COLUMN)
for j in range(num_channels):
print('{:10.5f}'.format(read_result.data[j]), 'V ',
end='')
#append a num_channels of data to the array (ROW)
myArray[new_index].append(read_result.data[i + j])
new_index+=1
print("\r")

            csvwriter.writerows(myArray) #Write the array to file
            csvfile.flush

        sleep(0.1)
```

```
    print('\n')
    csvfile.close()


if __name__ == '__main__':
    main()
```