MATLAB SCRIPT

User Documentation

By: Ziad Elraggal

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

Contents

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

Legend:

Black - is the unedited original MATLAB code that can be accessed and viewed in the script/ the related explanation provided for that section of code and how to troubleshoot it in case of any errors.

Green – is code to be provided to MATLAB.

Red – is the errors supplied by MATLAB during operation.

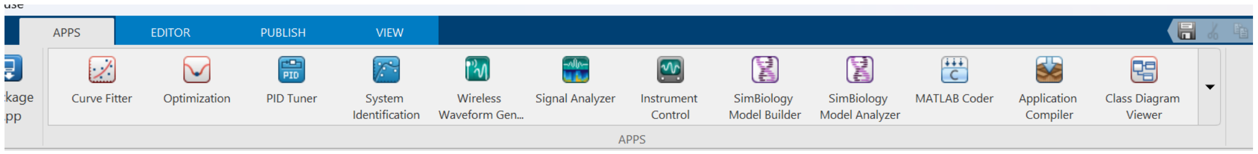
% Represents the start and end of an explanation related to a portion of code.

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

Installation & Setup:

To allow for communication between MATLAB and the Keithley 3706A, there are a few setup steps required:

1. Installing the Keithley IO Layer is software that includes required IVI and VISA software.
2. Install the Keithley 3700A IVI Driver.
3. Ensure MATLAB and all its components, specifically the **instrument control toolbox,** are installed (you can avoid this step if you have installed the full version of MATLAB. You can also ensure that the instrument control toolbox is installed by finding it in the apps section of MATLAB)

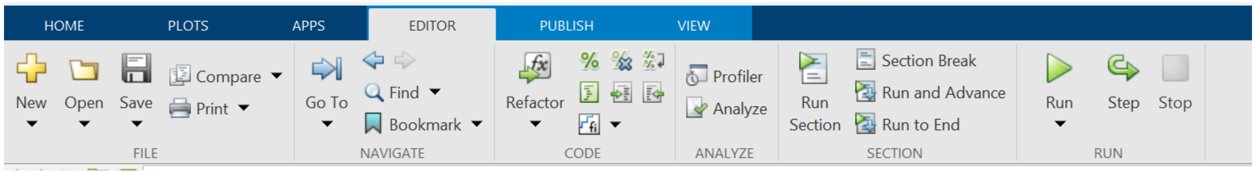


4. Copy and paste the script into a new MATLAB file and the test script into another new file for reasons specified in the next section.
5. You should be all set!

Getting Started:

The Keithly 3706A is a sensitive device that could encounter common issues/bugs. Below is a list of common bugs you may encounter while using this script with the machine and how to fix them. READ THIS SECTION BEFORE ATTEMPTING TO USE THE CODE.

1. Executing the script and halting it before reaching its intended conclusion: if, at any point in time, you stop the script using MATLAB's stop button on the top right menu, ***you will not be able to rerun the script without doing the following***. Navigate down to the console window containing this symbol ">>" and select it to type the following command "clear all." This command can also be run if you get the following error: "Creating a second device for the XXXXXXXXX resource is not supported. See related documentation for troubleshooting steps."
2. To avoid potential time waste, it is highly recommended to test the output of the device using the code in the Test Code section to ensure that the appropriate numerical values are being returned before running the required script.
3. If you encounter further issues, refer to the Keithly 3706A reference manual.
4. To start the program, press the live editor tab on the top menu and "Run".



Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

Original Code:

```
% Initialize parameters
obj = visadev("USB0::0x0XXX::0xXXXX::XXXXXX::0::INSTR");
obj.Timeout=20;
numTemps = 132;
Intervalmin = 1;
Durationhrs = 1;
Names = {'ex1', 'ex2', 'ex3', 'ex4', 'ex5', 'ex6', 'ex7', 'ex8', 'ex9', 'ex10',
'ex11', 'ex12'};
num = numel(Names);

% Generate point names
pointNames = cell(1, numTemps);
for n = 1:num
    for i = 1:11
        pointNames{(n-1)*11 + i} = [Names{n}, num2str(i)];
    end
end

% Initialize data storage
temperatureData = cell(1, 0);
timestamps = NaT(0, 1);

% Instrument setup
writeline(obj, 'reset()');

% Data collection loop
startTime = tic;
while toc(startTime) < (Durationhrs * 3600)
    i = length(temperatureData) + 1;
    x = datetime('now', 'Format', 'HH:mm:ss');
    temperatureMeasurements = measureTemperature(obj);

    temperatureData{i} = temperatureMeasurements;
    timestamps(i, 1) = x;

    minLength = min([length(pointNames), numel(temperatureData),
length(timestamps)]);
    numGraphs = ceil(numel(temperatureData) / 11);

    for graphIndex = 1:numGraphs
        figure;
        for n = 1:num
            subplot(3, 4, n);
            hold on;
            Indices = (n-1)*11 + 1:n*11;

            for i = (graphIndex-1)*11 + 1 : min(graphIndex*11, minLength)
                currentTimestamp = timestamps(i);

                if numel(temperatureData{i}) >= max(Indices)
                    color = hsv2rgb([n/num, 1, 1]);
```

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

```
                    scatter(repmat(currentTimestamp, 1, length(Indices)),
temperatureData{i}(Indices),"MarkerEdgeColor",color,"Marker", '*', 'DisplayName',
'');

                    for j = 1:length(Indices)
                        dataIndex = Indices(j);

                        if dataIndex <= numTemps
                            text(currentTimestamp, temperatureData{i}(dataIndex),
[pointNames{dataIndex}, '', ''], 'FontSize', 8, 'HorizontalAlignment', 'right');
                        end
                    end
                end
            end

            hold off;
            xlabel('Timestamp');
            ylabel('Temperature (Celsius)');
            title(['Temperature Measurements - ', Names{n}]);
            grid on;
        end

        pause(Intervalmin * 10);
        close(gcf);
    end
end

% Clean up
delete(obj);


% Determine minimum length for plotting
minLength = min([length(pointNames), numel(temperatureData), length(timestamps)]);
numGraphs = ceil(numel(temperatureData) / 11);

% Plotting loop
for graphIndex = 1:numGraphs
    figure;
    for n = 1:num
        subplot(3, 4, n);
        hold on;
        Indices = (n-1)*11 + 1:n*11;

        for i = (graphIndex-1)*11 + 1 : min(graphIndex*11, minLength)
            currentTimestamp = timestamps(i);

            if numel(temperatureData{i}) >= max(Indices)
                color = hsv2rgb([n/num, 1, 1]);
                scatter(repmat(currentTimestamp, 1, length(Indices)),
temperatureData{i}(Indices),"MarkerEdgeColor",color,"Marker", '*', 'DisplayName',
'');

                for j = 1:length(Indices)
                    dataIndex = Indices(j);
```

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

```
                    if dataIndex <= numTemps
                        text(currentTimestamp, temperatureData{i}(dataIndex),
[pointNames{dataIndex}, '', ''], 'FontSize', 8, 'HorizontalAlignment', 'right');
                    end
                end
            end
        end

        hold off;
        xlabel('Timestamp');
        ylabel('Temperature (Celsius)');
        title(['Temperature Measurements - ', Names{n}]);
        grid on;
    end
    saveas(gcf, strcat("TempGraph_", num2str(graphIndex), "_", datestr(now,
'HH_MM_SS_dd-mm-yyyy'), '.fig'));
end
%%
% Measurement function
function temperature = measureTemperature(obj)
    writeline(obj, 'channel.open("allslots")');
    writeline(obj, 'reading_buffer = dmm.makebuffer(1000)');
    writeline(obj, 'dmm.func = dmm.TEMPERATURE');
    writeline(obj, 'dmm.filter.type = dmm.FILTER_MOVING_AVG')
    writeline(obj, 'dmm.filter.count = 20')
    writeline(obj, 'dmm.filter.enable = dmm.ON')
    writeline(obj, 'dmm.nplc = 5');
    writeline(obj, 'dmm.transducer = dmm.TEMP_THERMOCOUPLE');
    writeline(obj, 'dmm.refjunction = dmm.REF_JUNCTION_INTERNAL');
    writeline(obj, 'dmm.thermocouple = dmm.THERMOCOUPLE_J');
    writeline(obj, 'dmm.units = dmm.UNITS_CELSIUS');
    writeline(obj, 'dmm.configure.set("mytemp")');
    writeline(obj, 'dmm.setconfig("1001:1040","mytemp")')
    writeline(obj, 'scan.create("1001:1040")')
    writeline(obj, 'scan.execute(reading_buffer)')
    responseString1 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    writeline(obj, 'dmm.setconfig("2001:2040","mytemp")')
    writeline(obj, 'scan.create("2001:2040")')
    writeline(obj, 'scan.execute(reading_buffer)')
    responseString2 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    writeline(obj, 'dmm.setconfig("5015:5035","mytemp")')
    writeline(obj, 'scan.create("5015:5035")')
    writeline(obj, 'scan.execute(reading_buffer)')
    responseString3 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    writeline(obj, 'dmm.setconfig("5015:5035","mytemp")')
    writeline(obj, 'scan.create("5015:5035")')
    writeline(obj, 'scan.execute(reading_buffer)')
    responseString4 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    %writeline(obj, 'dmm.setconfig("5001:5040","mytemp")')
```

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

```
    %writeline(obj, 'scan.create("5001:5040")')
    %writeline(obj, 'scan.execute(reading_buffer)')
    %responseString5 = writeread(obj, 'printbuffer(1, reading_buffer.n, 
reading_buffer)');
    %writeline(obj, 'dmm.setconfig("6001:6040","mytemp")')
    %writeline(obj, 'scan.create("6001:6040")')
    %writeline(obj, 'scan.execute(reading_buffer)')
    %responseString6 = writeread(obj, 'printbuffer(1, reading_buffer.n, 
reading_buffer)');
    responseString = append(responseString1, responseString2, responseString3, 
responseString4);

    valuesStr = strsplit(responseString);
    temperature = str2double(valuesStr);

    overflowThreshold = 9.900000000e+37;
    temperature(temperature == overflowThreshold) = NaN;
    invalidIndices = isnan(temperature);
    temperature(invalidIndices) = NaN;

    % Pad with NaNs to make the length divisible by 11
    remainder = mod(length(temperature), 11);
    if remainder > 0
        padding = 11 - remainder;
        for k = 1:padding
            temperature(end+1) = NaN;
        end
    end
end
```

Line-by-line Explanation:

```
% Initialize parameters
obj = visadev("USB0::0x0XXX::0xXXXX::XXXXXX::0::INSTR");
obj.Timeout=20;
```

% This line connects to the device in USB format. Depending on the Keithly 3706A, **REPLACE the information in quotation with the values supplied by MATLAB when you type the command *"visadevlist"*** (ensure the data you replace it with is also in quotations) in the command prompt

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

whilst the Keithly 3706A is connected to the computer being used. THIS IS ONLY NECESSARY IF THE SCRIPT RUNS INTO THE ERROR "Resource String is Invalid/not Found" or as a first-time setup. Furthermore, the next command increases the period until MATLAB stops the script because it takes too long to accommodate for longer NPLC, increasing data measuring accuracy. %

```
numTemps = 132;
Intervalmin = 1;
Durationhrs = 1;
temperatureData = cell(1, 0);
timestamps = NaT(0, 1);
```

% This code section contains variables that can be edited/modified depending on the circumstances. *"numTemps"* refers to the number of temperatures taken throughout the experiment. This value is arbitrary and is not used for more than reference in the code. *"Intervalmin"* refers to the duration of time between measurements. Edit this number freely to change the time period between measurements (this value is in minutes). *"Durationhrs"* refers to the duration of time this code runs for. Change this value freely to adjust the duration of the experiment (this value is in hours). *"temperatureData"*, *"timestamps," and "pointNames"* these three lines create an array and two cells to contain the measured data, their respective timestamps and the names of the individual thermocouples. DO NOT EDIT. %

```
% Instrument setup
writeline(obj, 'reset()');
```

% This line tells the machine to reset completely to prepare it for the script that will be run in the next few lines. Editing this line is unnecessary and could result in the script malfunctioning. %

```
startTime = tic;
while toc(startTime) < (Durationhrs * 3600)
    i = length(temperatureData) + 1;
    x = datetime('now', 'Format', 'HH:mm:ss');
    temperatureMeasurements = measureTemperature(obj);
```

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

```
    temperatureData{i} = temperatureMeasurements;
    timestamps(i, 1) = x;

    pause(Intervalmin * 60);
```

% This section of code initiates the while loop that will continuously acquire data until the specified time is met using the initially set value *“Duartionhrs”* set by the user. It also sets the interval between measurements using *“Intervalmin”,* which was initially set by the user. Finally, it contains a code set to collect the data from the device using a function and creating timestamps for the data points collected. There is no reason to edit this section at all. %

```
    minLength = min([length(pointNames), numel(temperatureData), 
length(timestamps)]);
    numGraphs = ceil(numel(temperatureData) / 11);
for graphIndex = 1:numGraphs
        figure;
        for n = 1:num
            subplot(3, 4, n);
            hold on;
            Indices = (n-1)*11 + 1:n*11;

            for i = (graphIndex-1)*11 + 1 : min(graphIndex*11, minLength)
                currentTimestamp = timestamps(i);

                if numel(temperatureData{i}) >= max(Indices)
                    color = hsv2rgb([n/num, 1, 1]);
                    scatter(repmat(currentTimestamp, 1, length(Indices)), 
temperatureData{i}(Indices),"MarkerEdgeColor",color,"Marker", '*', 'DisplayName', 
'');

                    for j = 1:length(Indices)
                        dataIndex = Indices(j);

                        if dataIndex <= numTemps
                            text(currentTimestamp, temperatureData{i}(dataIndex), 
[pointNames{dataIndex}, '', ''], 'FontSize', 8, 'HorizontalAlignment', 'right');
                        end
                    end
                end
            end

            hold off;
            xlabel('Timestamp');
            ylabel('Temperature (Celsius)');
            title(['Temperature Measurements - ', Names{n}]);
            grid on;
        end
```

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

```
        pause(Intervalmin * 10);
        close(gcf);
    end
end
```

%This section of code continuously plots the data for every loop to allow for monitoring during the process. The data is displayed in a format and graph similar to the results. These graphs are, however, displayed and then deleted between each interval as they are only meant for monitoring purposes. As the data is plotted using timestamps, the right-most data set is the newest set as it is the most recent. %

```
delete(obj);

% Plotting loop
for graphIndex = 1:numGraphs
    figure;
    for n = 1:num
        subplot(3, 4, n);
        hold on;
        Indices = (n-1)*11 + 1:n*11;

        for i = (graphIndex-1)*11 + 1 : min(graphIndex*11, minLength)
            currentTimestamp = timestamps(i);

            if numel(temperatureData{i}) >= max(Indices)
                color = hsv2rgb([n/num, 1, 1]);
                scatter(repmat(currentTimestamp, 1, length(Indices)),
temperatureData{i}(Indices),"MarkerEdgeColor",color,"Marker", '*', 'DisplayName',
'');

                for j = 1:length(Indices)
                    dataIndex = Indices(j);

                    if dataIndex <= numTemps
                        text(currentTimestamp, temperatureData{i}(dataIndex),
[pointNames{dataIndex}, '', ''], 'FontSize', 8, 'HorizontalAlignment', 'right');
                    end
                end
            end
        end

        hold off;
        xlabel('Timestamp');
        ylabel('Temperature (Celsius)');
```

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

```
        title(['Temperature Measurements - ', Names{n}]);
        grid on;
    end
    saveas(gcf, strcat("TempGraph_", num2str(graphIndex), "_", datestr(now,
'HH_MM_SS_dd-mm-yyyy'), '.fig'));
end
```

% This snippet of code is responsible for deleting the object created to connect to the device (otherwise, the code can not be rerun as highlighted in the before you start section). It is also responsible for creating the scatter sub-plots containing the names specified earlier, axis labels, and graph titles, with a new plot being created for each REDACTED to simplify data reading further (keep in mind that this is done for every thermocouple assuming there is 11 per sample). Finally, it saves the graph as a figure called "TempGraph," followed by the appropriate timestamp. There is no need to edit this section of code unless you wish to change the axis labels or title, which can be done by editing the name between the quotation in the *"xlabel"*, *"ylabel"*, or *"title"* commands. Finally, this snippet saves the graph as a .fig file with the name *"TempGraph"* as well as the time and today's date. %

```
% Measurement function
function temperature = measureTemperature(obj)
    writeline(obj, 'channel.open("allslots")');
    writeline(obj, 'reading_buffer = dmm.makebuffer(1000)');
    writeline(obj, 'dmm.func = dmm.TEMPERATURE');
    writeline(obj, 'dmm.filter.type = dmm.FILTER_MOVING_AVG')
    writeline(obj, 'dmm.filter.count = 20')
    writeline(obj, 'dmm.filter.enable = dmm.ON')
    writeline(obj, 'dmm.nplc = 5');
    writeline(obj, 'dmm.transducer = dmm.TEMP_THERMOCOUPLE');
    writeline(obj, 'dmm.refjunction = dmm.REF_JUNCTION_INTERNAL');
    writeline(obj, 'dmm.thermocouple = dmm.THERMOCOUPLE_J');
    writeline(obj, 'dmm.units = dmm.UNITS_CELSIUS');
    writeline(obj, 'dmm.configure.set("mytemp")');
    writeline(obj, 'dmm.setconfig("1001:1040","mytemp")')
    writeline(obj, 'scan.create("1001:1040")')
    writeline(obj, 'scan.execute(reading_buffer)')
    responseString1 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    writeline(obj, 'dmm.setconfig("2001:2040","mytemp")')
    writeline(obj, 'scan.create("2001:2040")')
    writeline(obj, 'scan.execute(reading_buffer)')
    responseString2 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    writeline(obj, 'dmm.setconfig("5015:5035","mytemp")')
    writeline(obj, 'scan.create("5015:5035")')
    writeline(obj, 'scan.execute(reading_buffer)')
    responseString3 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    writeline(obj, 'dmm.setconfig("5015:5035","mytemp")')
    writeline(obj, 'scan.create("5015:5035")')
    writeline(obj, 'scan.execute(reading_buffer)')
```

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

```
    responseString4 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    %writeline(obj, 'dmm.setconfig("5001:5040","mytemp")')
    %writeline(obj, 'scan.create("5001:5040")')
    %writeline(obj, 'scan.execute(reading_buffer)')
    %responseString5 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    %writeline(obj, 'dmm.setconfig("6001:6040","mytemp")')
    %writeline(obj, 'scan.create("6001:6040")')
    %writeline(obj, 'scan.execute(reading_buffer)')
    %responseString6 = writeread(obj, 'printbuffer(1, reading_buffer.n,
reading_buffer)');
    responseString = append(responseString1, responseString2, responseString3,
responseString4);

    valuesStr = strsplit(responseString);
    temperature = str2double(valuesStr);

    overflowThreshold = 9.900000000e+37;
    temperature(temperature == overflowThreshold) = NaN;
    invalidIndices = isnan(temperature);
    temperature(invalidIndices) = NaN;

    % Pad with NaNs to make the length divisible by 11
    remainder = mod(length(temperature), 11);
    if remainder > 0
        padding = 11 - remainder;
        for k = 1:padding
            temperature(end+1) = NaN;
        end
    end
end
```

% This section of the code refers to setting up the device to take the measurements required. It starts by opening all channels on the device to allow editing of individual channel settings. The code then continues to create a buffer to store the collected data, set the function to temperature, create a filter to increase the accuracy of measurements, and prevent electrical noise(according to your accuracy requirements, change this value between 1-100, a lower value means a larger change is occurring in temperature between measurements while a higher value means that your temperature is most likely stable) , sets NPLC to 5 further increase accuracy, sets the measurement device to the thermocouple and the thermocouple type to *"J"*,(you can adjust this value depending on the type of thermocouple used) it then sets the units to Celsius, (can be changed to *"FAHRENHEIT"* or *"KELVIN"* typed in that exact way) names the configuration to *"mytemp"* (arbitrary name), sets that configuration to multiple channels, (this is the only section that should be edited depending on the task it has been preset to accommodate the

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

required four channels. However, the ***Green*** highlighted part of the code represents ***commented*** code, meaning that it will NOT BE executed until the *"%"* sign is removed at the start of the line. This should only be done if you utilize channel slots 5 and 6 (if this is the case change the line *"responseString = append(responseString1, responseString2, responseString3, responseString4);"* to *""responseString = append(responseString1, responseString2, responseString3, responseString4, responseString5, responseString6*);" if you are using all available slots otherwise if you are only using either 5 or 6 remove the corresponding *"reponsestringX"* line keep in mind the X only refers to the sequential ordering of the code ***NOT*** the actual slots numbers this is only important if you switch around the slots and are not using them sequentially). Otherwise, if you wish to use another channel combination, e.g., 1 3 5 6, then proceed to change the first numbers sequentially. e.g. 1001:1040 next set of numbers would be 3001:3040 instead of 2001:2040 and so on. To know which range of channels to access, the first number, '1' in this case, is the slot in which the card is connected on the back of the device. The next string of numbers between 1 and 40 refers to the channels to which the thermocouples are connected on the actual external card ((the reason for this part of the code is that channels 41 and 42 ***cannot*** be utilized for temperature measurements, so to prevent an error in the naming scheme, ensure the values are always X001:X040). These numbers can be read by looking at the physical card and its junctions) and then creating a scan with that **SAME** set of channels. **ENSURE** the string of numbers matches (i.e., the immediate next string of numbers, ***not all numbers***); otherwise, no data will be measured across said channels. Finally, the data is read from the buffer and stored to be plotted. The last few lines of code should not be edited as they are used to allocate data types to the acquired data and prevent overflow errors from being included in the final graph. Finally, it also adjusts the data by adding "padding" if the number of tFohermocouples used is not divisible by 11 to allow for appropriate plotting. %

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.

Foot Notes:

The card required to use this software is a **Keithley 3721 Dual 1x20 Multiplexer Card** with its respective accessory card **Keithley 3721-ST Screw Terminal Block**. The code, in theory, is usable with any card that allows for thermocouples reading with a screw terminal block attached, but the channels and other settings might differ. Please follow the replacement card files and change the code accordingly.

Test Code:

```
obj=visadev("USB0::0x05E6::0x3706::04076871::0::INSTR");
writeline(obj,'reset()')

writeline(obj,'reading_buffer=dmm.makebuffer(1000)')
writeline(obj,'dmm.func = dmm.TEMPERATURE')
writeline(obj, 'dmm.filter.type = dmm.FILTER_MOVING_AVG')
writeline(obj, 'dmm.filter.count = 100')
writeline(obj, 'dmm.filter.enable = dmm.ON')
writeline(obj,'dmm.nplc=10')
writeline(obj,'dmm.transducer=dmm.TEMP_THERMOCOUPLE')
writeline(obj,'dmm.refjunction=dmm.REF_JUNCTION_INTERNAL')
writeline(obj,'dmm.thermocouple=dmm.THERMOCOUPLE_J')
writeline(obj,'dmm.units=dmm.UNITS_CELSIUS')
writeline(obj,'dmm.configure.set("mytemp")')
writeline(obj,'dmm.setconfig("XXXX","mytemp")')
writeline(obj,'scan.create("XXXX")')
writeline(obj,'scan.measurecount=1')
writeline(obj,'scan.execute(reading_buffer)')
CIDC1=writeread(obj,'printbuffer(1,1,reading_buffer)');
disp(CIDC1)
delete(obj)
```

Note: If the code is edited at any point to facilitate new tasks, please maintain the original code in a separate file and add notes to this documentation to allow ease of communication and usage for the next user.