Keithley 2281-20-6 .dll file – Instrument Handling through USB Communication

# Prerequisites

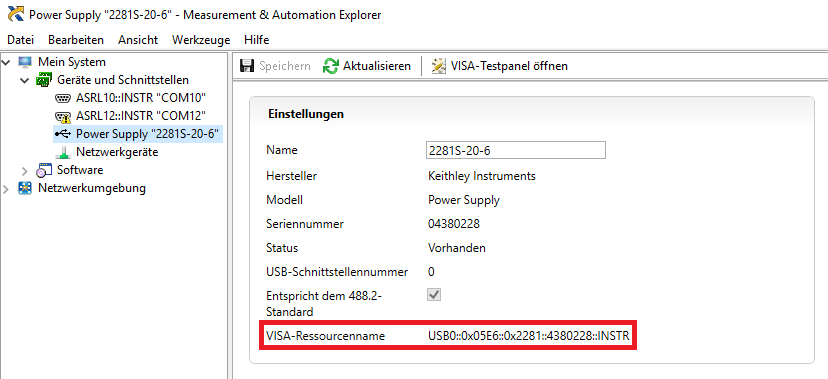
All the prerequisite files and executables stated below are included in the Folder “Drivers and Libraries“.

1. IVI Shared Components

DCPwr, Dmm, Driver and VISA Libraries are used by the .dll file.

1. National Instruments VISA

The VISA library is used by the .dll file. Furthermore, the NI MAX application is necessary to determine the VISA resource name of the Keithley 2281S-20-6 instrument, as shown below.



1. Keithley 2280 driver

The driver is necessary in order to communicate succesfully with the Keithley 2281S-20-6 instrument.

1. Relay Card Driver

The driver is necessary in order to communicate succesfully with the Conrad Relay Card (197720).

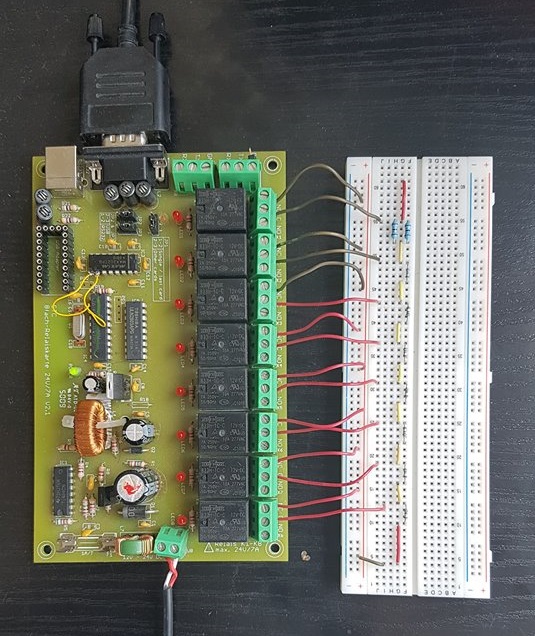
1. USB – RS232 Communication driver

The driver is necessary in order to establish a succesful communication via the USB to RS232 cable.

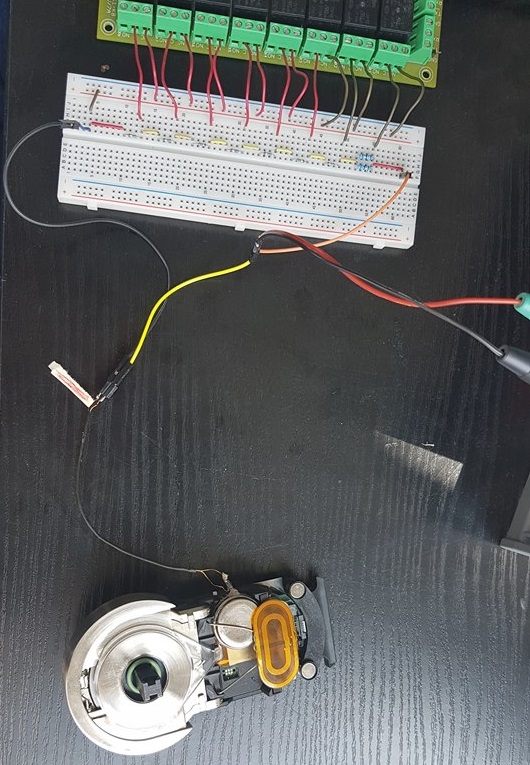
# Assembly of Keithley 2281 – Relay Card – Smart Lock

The circuit assembly of all devices is shown in the pictures below.

1. Relay Card – Circuit Board with 10 Ohm resistors



1. Assembly of all devices



# Documentation

As far as the Power Supply Mode is concerned, Keithley has created and freely distributed the KickStart 2 Software, which is also included in the Folder “Kick Start Software“.

#region Class Keithley2281:

* Constructor public Keithley2281(string resourceName): This method receives the resource name of the instrument and tries to establish connection with the device (Device should be powered on). If the connection is established, then the output of the instrument is disabled and a message with the instrument name is displayed on the console. Afterwards, this function calls the FindRelayCard method, in ordert o establish communication with the Relay Card as well. If the connection fails, an error message is displayed and the program ends.
* public void Exit(): This method is responsible to end the operation of the programm. To do so, it disables the output and clears the data buffer of the Keithley 2281S-20-6 instrument, switches off all relays of the Relay Card, disconnects from the relay Card, returns to the Home Screen and finally disconnects from the Keithley 2281S-20-6 instrument as well.
* private void EnableOutput(): This method enables the output of the instrument, regardless of the operation mode. By enabling the output, the buffer starts saving measurements.
* private void DisableOutput(): This method disables the output of the instrument, regardless of the operation mode. By disabling the output, no new measurements are made.
* public void ChooseOperationMode (string operationMode): This method receives the desired operation mode in a string value and then calls the corresponding method to initialize the operation mode. Available string values are: Power Supply, Battery Test, Battery Simulator and Close Connection. The Close Connection value terminates the current connection with the instrument and the program ends.
* public int ChooseChannel (int channel): This method requests from the user to decide between the two available instrument channels.
* public void CloseConnection(): This method terminates the current connection with the instrument and returns a corresponding message.

#region Simple Help Functions – Check Variable Content

* public void CheckRange (ref int val, int margin1, int margin2): This method delimits the received integer value between the given boundaries.
  + overload (ref double val, double margin1, double margin2): As above, but for double values.

#region Error Handling

* public void DeviceErrors(): This method accesses the instrument’s error stack and returns the error code(s) and message(s) on the console.

#region General Purpose Setters, Getters, Settings and Configuration Methods

This region contains public setters ( SetValName(valType val) ) and getters (GetVal) for the following parameters:

* Overvoltage Protection (OVP): Range for this variable is [0, 21] V
* Overcurrent Protection (OCP): Range for this variable is [0.1, 6.1] A
* Maximum Voltage Protection (MAXV): Range for this variable is [0, 21] V
* Power Line Frequency (PLF): Read Only – Keithley2280PowerLineFrequency Enum (50 or 60)
* Sample Interval (SampleInterval): for 50 Hz PLF – [0.00008, 0.48] seconds, for 60 Hz PLF – [0.000066667, 0.5]. Can set only for Battery Test and Simulator modes.
* Trigger:
  + State (TriggerState): boolean variable ON(1) / OFF(0)
  + Arm Count (ArmCount): Range for this variable is [1, 2500]
  + Trigger Count (TriggerCount): Range for this variable is [1, 2500]
  + Trigger Source (TriggerSource): Values range for this variable is Keithley2280TriggerSourceEnum Bus, External, Immediate, Manual. This method is same for both Trigger and Arm modes.
  + Sample Count (SampleCount): Range for this variable is [1, 2500]
  + List Type (TriggerListType): Values range for this variable is Keithley2280ListMCompleteEnum Step or State – only available in Power Supply mode.

#region Battery Test Operation Code

* public void BatteryTestMode(): Initializes the Battery Test operation mode. This is the only method needed to be used in the main program in order to initiate a full AH Measurement.
* public void ESRMeasurement (ref double evocDelay): This method is used to get a single, instant measurement of the internal resistance (ESR) and the Open Circuit Voltage of the Battery. The evocDelay parameter represents the amount of time for the instrument to wait before it resumes the measurement. evocDelay range: [0, 1200000] seconds. After the ESR and Voc values are measured, output is turned off.

#region AH Measurement and Battery Model Generation

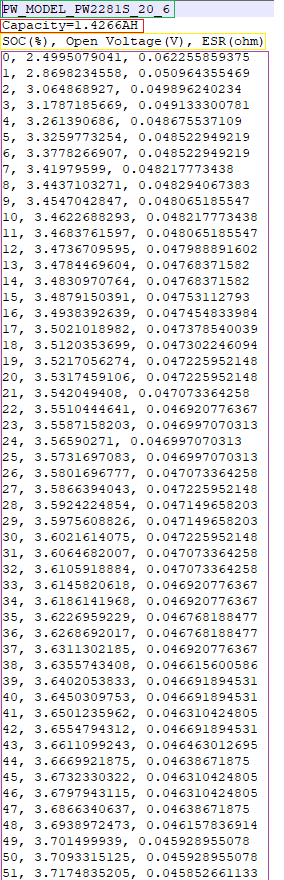
* public void SelectESRInterval (ref Keithley2280BatteryTestMeasureESRIntervalEnum esrInterval): This method is responsible for adjusting the time interval between consecutive measurements during an AH measurement Test. Available options are: 10 second, 30 seconds, 60 seconds, 120 seconds and 10 minutes.
* private void FullyDischargeBattery (ref double vLevel, ref double endCond): This method receives the Voltage Level and End Condition of the Battery to be discharged, checks whether those values meet the corresponding criteria and then enables the output to start the discharge of the battery. After the Battery is fully discharged, the SelectESRInterval() method is called in order to define the sample interval to be used in the following AH Measurement Test. After the interval is adjusted, the AHMeasurement() method is called to start the test.
* private void AH Measurement (ref double fullV, ref double iLimit, Keithley2280Battery TestMeasureESRIntervalEnum interval): This method is used to initiate an AH Measurement Test. Firstly, the method evaluates the given values for Full Voltage, Current Limit and Sample Interval by calling the CheckRange() method. Afterwards, the AH Measurement State is changed to Start and a confirmation message is displayed on the console.
* public void GenerateBatteryModel (ref double minVOC, ref double maxVOC, ref int index): This method receives the minimum and maximum Open Circuit Voltages of a Battery Model, checks their validity, creates a new Battery Model and saves it on the instrument’s memory.

#region Battery Simulator Operation Code

#region Inititate and Close Battery Simulator

* private void BatterySimulatorMode(): This method is used to initiate the Battery Simulation operation mode. It checks for connection errors and sets the instrument display screen on the Battery Simulator home screen. This method has 3 overload methods, which use their parent function for error handling purposes:
  + public void BatterySimulatorMode (int index): This method recalls a Battery model, which is already stored in one of the 9 memory spaces allocated by the instrument‘s internal memory for for Battery Models.

* + public void BatterySimulatorMode (Keithley2280BatteryBuildInModelEnum model): This method recalls one of the 5 built-in models provided by the instrument. These available models are: 12 Volts Nickel Metal Hydride Battery, 1 Point 2 Volts Nickel Cadmium Battery, 1 Point 2 Volts Nickel Metal Hydride Battery, 4 Points 2 Volts Lithium Ion Battrey and Lead Acid Battery.
  + public void Battery Simulator Mode (string filepath): This method receives a string with a filepath to a .csv file, which must contain the necessary data for a Battery Model to be generated and stored inside the intrument’s memory storage. Following is an example of the contents of such a file.

 **Notes:**

- The first line of the file must contain the Battery Model name.

- The second line of the file must contain the Capacity value of the Battey. The format, as shown, should be “Capacity=X.YYYYAH“.

- The third line should contain a string with the same order, as shown in the image.

- From the fourth line and downwards, the lines should contain the SOC (in percentage level), the Open Circuit Voltage (in Volts) and the internal resistance of the Battery at the corresponding level of charge, seperated with a comma. If the Battery Model should be coarsely accurate, then 11 lines should be stored (for SOC = 0, 10, 20, … , 100). If the Battery Model is finely accurate, then 101 lines should exist inside the .csv file.

Tabelle 1: Correct Battery Model File Example (.csv file)

#region Variable Handling, Setters and Getters

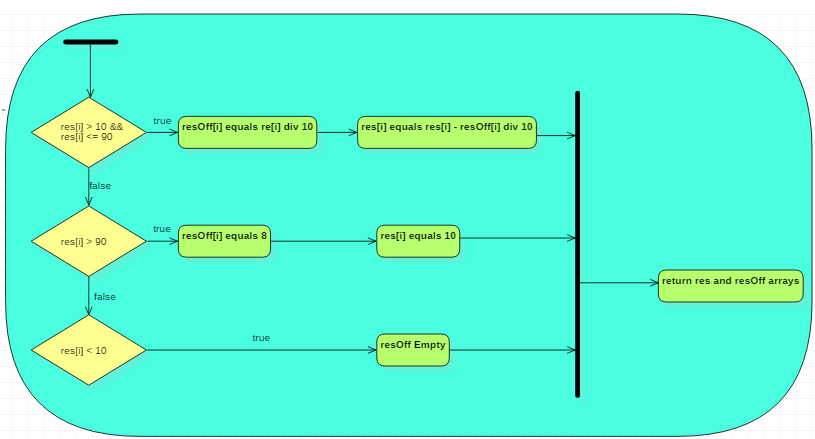
This region contains public setters ( SetValName(valType val) ) and getters (GetVal) for the following parameters:

* Method (Method): Static or Dynamic Battery Simulation Mehtod.
* Open Circuit Voltage (VOC): Range for this variable is [EmptyV, FullV] V.
* State of Charge (SOC): Range for this variable is [0, 100].
* Full Voltage (FullV): Open Circuit Voltage at 100% Battery level.
* Empty Voltage (EmptyV): Open Circuit Voltage at 0% Battery level.
* Current (Current): Returns an instant Current Measurement. Read Only.
* Current Limit (ILimit): Range for this variable is [0.1, 6.1] A.
* Capacity (Capacity): Returns an instant Capacity Measurement. Read Only.
* Capacity Limit (CapacityLimit): Range for this variable is [0.001, 99] AH.
* Resistance (Resistance): Read Only - Range for this variable is [0, 10] Ohm
* Resistance Offset (ResistanceOffset): Range for this variable is [-100, 100] Ohm. Cannot be changed while simulation is in progress.
* Status (Status): Read Only – Values for this variable are Charging or Discharging
* Terminal Voltage (TerminalVoltage): Returns an instant Terminal Voltage Measurement. Read Only.

#region Battery Model Handling

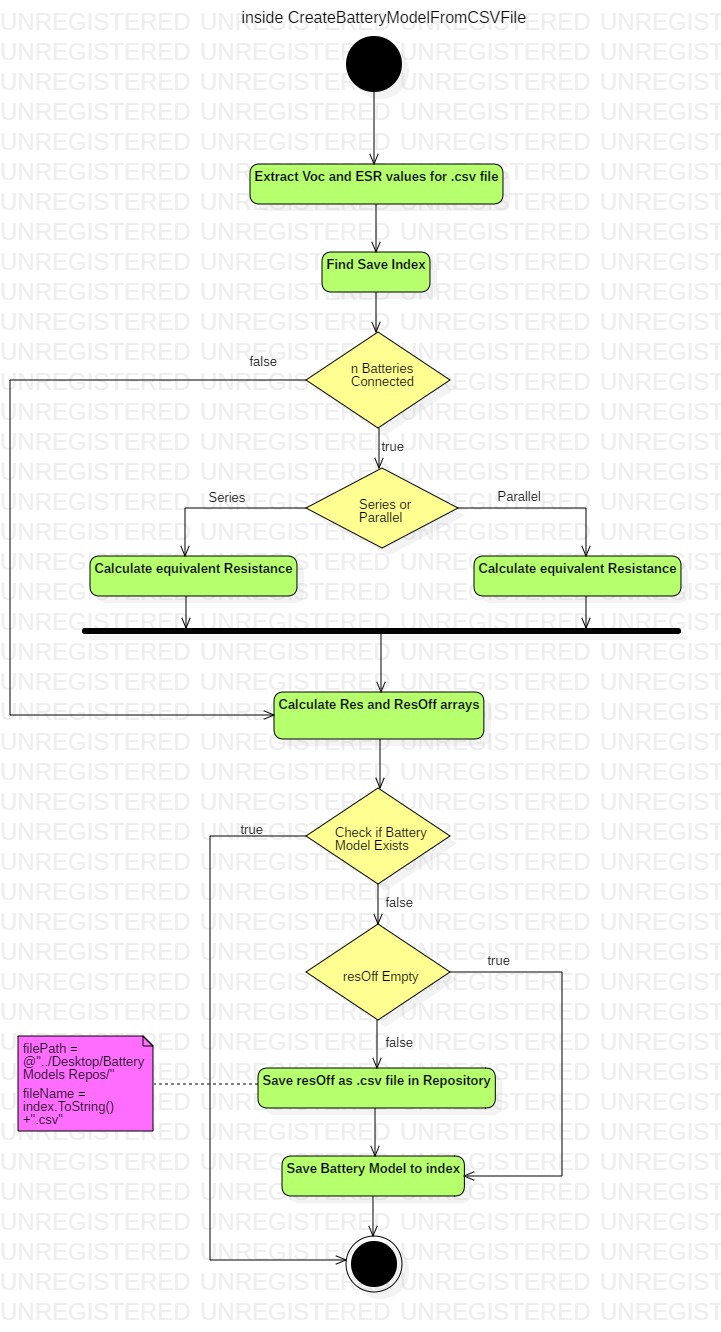
* private bool CheckIndexContent (int index): Checks if the assigned Battery Model index of the Instrument is empty (returns true) or not.
* private void CheckFileContent (ref string filePath): Checks whether the file containing the battery model to be configured by the instrument is of type .mdl, .csv and .txt. If not, an error message is displayed. Otherwise, the method responsible for the import of the battery model to the instrument is called. If the filePath is wrong, then this method asks the user whether he wants to type another filePath. If not, all available (saved and built-in) Battery Models are displayed on the console and the program waits for the user to select one of them. An abort option (=0) is also available.
* private int FindEmptyIndex(): Returns first available (=empty) Battery Model Index of the Instrument. If all Battery Model indexes have already been stored with models, the method returns the value 0.
* private void FindResistanceOffset(int number): This method checks whether a Resistance Offset File exists on the Repository and creates a new resOffset array containing the correct values. If no such file exists, the resOffset array iss et to have zero values for 101 indexes.
* private Keithley2280BatteryModelTypeEnum FineOrCoarse (int length): This method returns a Keithley2280BatteryModelTypeEnum with a value which determines whether the tuning of the Battery Model is Fine (length = 101) or Coarse (length = 101) by examining the assigned Array’s Length value (passe as the arg length).
* private int FineOrCoarse (Keithley 2280 Battery Model Type Enum): This method returns the length of the Open Circuit Voltage (VOC) and Internal Battrey Resistance (ESR) arrays of the desired Battery Model, according to whether the tuning of the model is Fine or Coarse.
* private bool CheckIfBatteryModelExists (double[] voc, double[] res, double cap): Returns whether the desired Battery Model (args are Open Circuit Voltage, Resistance, Capacity and Resistance Offset of the desired Battery Model) exists and is already saved on the Instrument (=true) or not. For the Resistance Offset parameter, the program searches for a .csv file with all the stored Resistance Offset values for each one of the 9 available index spaces on the Instrument. For example, for index number 4, the filepath to this file must be “C:\user\................\Desktop\Keithley 2281\Battery Models Repository\ResistanceOffset4.csv“. If the Battery Model already exists, then it is recalled and set as active on the Keithley 2281S-20-6 instrument.
* public void LoadBuiltInModel(): Loads one of the five already built-in models of the instrument. Returns error message if the index input is false. Has option to abort operation.
* public void LoadBuiltInModel (Keithley2280BatteryBuildInModelEnum model): Loads one of the five already built-in models of the instrument through the integer argument given, without asking for user input. Input range should be [10, 14]. If the imput is wrong, then the original LoadBuiltInModel is called, which displays the available built in Batery Models on the screen and waits for the corresponding user input.
* private int[] SeparateResAndResOffsetArrays (ref double[] res): This method is responsible for the correct distribution of the Resistance values. Due to the instrument’s limitation to a maximum of 10 Ohm of internal Resistance (ESR) values for each Battery Model and State Of Charge, values that are higher than 10 Ohm must be treated differently. The solution to this problem is given via a relay card, which has 8 ports, each connected to a 10 Ohm resistor and all together connected in series. This means that the instrument can cover up to 90 Ohm of Internal Battery Resistance (ESR) in total, through the Relay Card.

This method uses the following algorithm to separate the Resistance values given to the instrument and the ones given to the Relay Card.



After the calculations, the Resistance array is updated with the values which are bound to be stored to the Instrument’s internal memory. Furthermore, a new array is created and returned to the calling double[,] array. This is the Resistance Offset array, which contains tha values of the resistance to be produced by the relay card for each battery percentage, as well as, the nuber of relays used at each percentage.

* private void BatteriesSeries (int n, ref double[] voc, ref double[] res): This method is called, when the user requires n batteries of the same battery model to be connected in series. The arguments included are the number of connected batteries (given by the user through the console), the Open Circuit Voltage and total Resistance Arrays of the Battery Model used. After the necessary calculations, the Open Circuit Voltage and Resistance Arrays are updated.
* private void BatteriesSeriesReverse (int n, ref double[] voc, ref double[] res): This method is called, when the user requires to extract the Battery Model of one Battery from a model, where n batteries of the same battery model were connected in series. The arguments included are the number of connected batteries (given by the user through the console), the Open Circuit Voltage and total Resistance Arrays of the Battery Model used. After the necessary calculations, the Open Circuit Voltage and Resistance Arrays are updated.
* private void BatteriesParallel (int n, ref double[] res, ref double cap): This method is called, when the user requires n batteries of the same battery model to be connected in parallel. The arguments included are the number of connected batteries (given by the user through the console), the total Resistance Array and the Capacity of the Battery Model used. After the necessary calculations, the Resistance Array and the Capacity value are updated.
* private void BatteriesParallelReverse (int n, ref double[] res, ref double cap): This method is called, when the user requires to extract the Battery Model of one Battery from a model, where n batteries of the same battery model were connected in parallel. The arguments included are the number of connected batteries (given by the user through the console), the total Resistance Array and the Capacity of the Battery Model used. After the necessary calculations, the Resistance Array and the Capacity value are updated.
* Public void CreateBatteryModelFromCSVFile (System.IO.Streamreader file): This method gets input (as a Stream) from a .csv file and creates a new Battery Model which will then be stored in the device’s internal memory. After the file content is read, the console prompts a message to the user whether he wants more Batteries of the same model to be connected either in Series or in Parallel with one another. Then, the mehtod queries the already existing Battery Models saved on the instrument and checks if the desired model already exists. If not, it searches for any empty indexes (by calling the findEmptyIndex method) and if there is any, then it stores the model in the empty index. If no index is empty, it prompts the user to either select an index to be overwritten or abort the save process. Due to a limitation of the instrument, resistance values higher than 10 Ohm cannot be handled by the instrument alone. As mentioned above (SeparateResAndResOffsetArrays method), the values higher than 10 Ohm are stored in a new file and handled through a Relay Card. A more remonstrant display of this method is shown below.



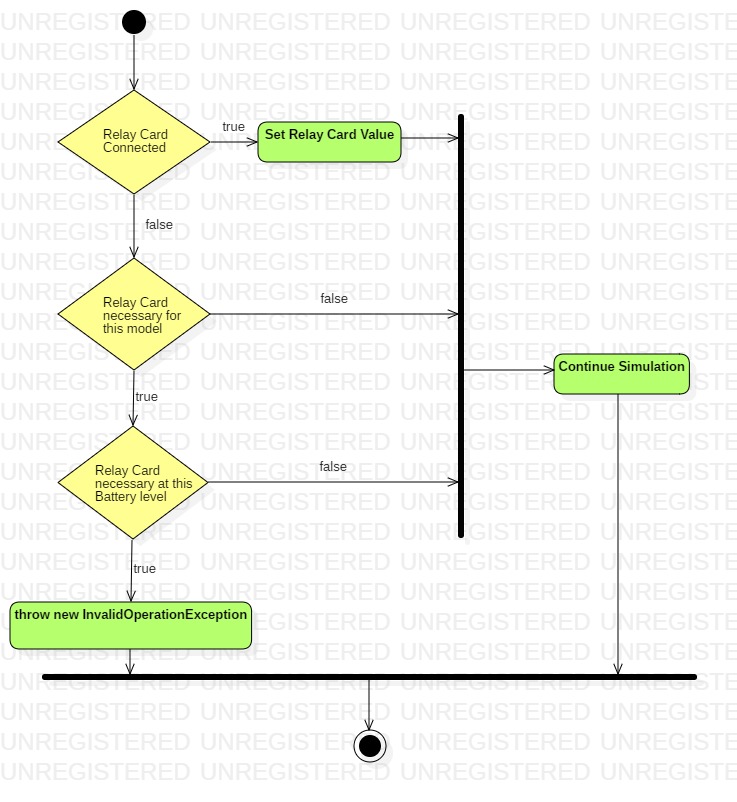
* private void ShowSavedBatteryModels(): This method displays all saved Battery Models on the intrument’s memory. For the saved Battery Models, the Full and Empty Open Circuit Voltages are displayed, as well as the Battery Capacity. As for the built-in models, their Battery Model names are displayed on the console.
* private void LoadBuiltInModel (int battModel): Loads one of the five already built-in models of the instrument through the argument given, without asking for user input.
* public void SaveBatteryModel (int index): This method queries a Battery Model (passed by through the argument) from the ones saved on the instrument’s memory and creates a .csv file with the data (Voc, ESR and Capacity) of the Battery Model to be saved on your computer. The filepath to this file will be “C:\user\...\Desktop\Keithley 2281\Battery Models Repository\Model4.csv“

#region Simulation

* public void Start Simulation(): The main method to start a battery simulation. This method handles the Relay Card, sets the correct relays at work, clears the buffer contents and enables the output of the device, so as to start the simulation measurements. After the simulation is over, this method invokes the necessary methods to store the measurements in a .csv file and exits the Battery Simulation Mode. The configuration settings used are the ones used during the last simulation performed by the instrument. To adjust these settings before the simulation, one could use the bleow mentioned overloads of this method:
  + - StartSimulation(Keithley2280BatterySimulationModeEnum simMode): Set the Simulation Mode (Static/Dynamic) before the simulation starts.
    - StartSimulation(double soc): Set the desired percentage of Battery level before the simulation starts.
    - StartSimulation(double soc, double sampInt): Set the desired percentage of Battery energy and the sampling interval of the measurements before the simulation starts.
    - StartSimulation(Keithley2280BatterySimulationModeEnum simMode, double soc): Set the Simulation Mode (Static/Dynamic) and the desired percentage of Battery energy before the simulation starts.
    - StartSimulation(Keithley2280BatterySimulationModeEnum simMode, double soc, double sampInt): Set the Simulation Mode (Static/Dynamic), the desired percentage of Battery energy and the sampling interval before the simulation starts.

#region Relay Card Handling

* public void FindRelayCard(): This method searches at all connected COM ports for the Relay Card. In order to do so, an initialization command (according to the Relay Card manual) is sent to each device. If the response is correct, then the COM port value is stored and the conntection with the Relay card is established. If no Relay Card is found, then a corresponding message is displayed on the console.
* private void SwitchOffRelay(): This method is used to switch off all relays on the Relay Card.
* private int LogicalXOR (int op1, int op2): This method performs the Logical XOR operation given 2 input numbers.
* private int LogicalXOR (int op1, int op2, int op3): This method performs the Logical XOR operation given 3 input numbers. Both LogicalXOR methods are used to calculate the input bytes of the Relay Card.
* private void SetRelayCardValue(): This method is responsible for setting on/off all the relays of the Relay Card., according to the resOffset array. If no Relay Card is connected, then the following algorithm is used in ordert o determine whether and how the simulation can continue.



#region Save Data and Export Measurements

* private void SaveResistanceOffsetToRepository(): This method gets the Resistance Offset Array and creates a new fileStream, which is then passed on to the StoreDataToCSVFile() in order to be saved on your computer. This method is always called when a new Battery Model is inserted and the Resistance Offset Array is not empty. By doing so, the main program is able to handle the Relay Card, as well as query the Battery Models, so that no duplicate models exists.
* public List<string> Save Data(): This method is used to export the data from the device buffer during the Battery Simulation Mode and save it on a List. The Keithley driver does not provide a direct method to fetch the data from the buffer during the Battery Simulator operation mode, so SCPI commands are used for the communication with the device. Measurements of voltage, current, state of charge, resistance and timestamp are stored in each line of the list, seperated with a comma. The resistance value has accuracy of 2 digits inside the buffer, which is not enough for many battery models. To get 4-digit accuracy one must send a command to the device to get the instant value of the resistance of the battery. This measurement is not concurrent with the other measurements inside the buffer index. The time difference corresponds to the time needed to execute 2 consecutive commands inside the .dll file (≈ 23 ms). This method also handles Dynamic Simulations, meaning that it is also able to change the state of the relays on the Relay Card, according to the Battery level. The simulation continues until the user presses any button.
* public void StoreDataToCSVFile (List<string>): This method is used to create and save .csv files on the computer’s local disk. It used to create files with Battery Models, Resistance Offset Arrays or measurement data produced during the Battery Simulation Mode. The following table shows the names of the newly created files and the directories where they are about to be saved on.

|  |  |  |
| --- | --- | --- |
| **Containing Data** | **Directory** | **File Name** |
| Battery Model | C:\user\..\Desktop\Keithley 2281\Battery Models Repository | “ModelX.csv“ where X = index number |
| Resistance Offset Array | C:\user\..\Desktop\Keithley 2281\Battery Models Repository | “ResistanceOffsetX.csv“ where X = index number |
| Simulation Data | C:\user\..\Desktop\Keithley 2281\Simulation Data | “Test\_dd\_mm\_yy\_\_HHh\_MMm.csv“  Where dd = day, mm = month, yy = year, HH = hours and MM = minutes |

# Code Example

Some source code is provided below, with the intention to help the integration of the .dll file to another project. The use of try/catch statements is recommended for better handling of unexpected exceptions and errors.



This code is also provided in the Folder “Repos/Code Example“.

# Experimental / Simulation Data

Some Simulation data, as well as some already provided/created under the Folders “Keithley 2281/Battery Models Repository“ and “Keithley 2281/Simulation Data“.