CAN AUTOMATION  
PYTHON SCRIPT

**Overview**

Before delving into the detailed explanation of the Python script, it's important to provide some context. This script is specifically designed to operate within the PyCharm environment, facilitating interaction with a Roboteq controller via CANOpen through a dedicated DLL named "RoboLabViewUtils". Currently, the script is undergoing its initial development stage, aiming to establish a robust foundation for future expansion.

This initial version is focused primarily on firmware V3.1, although provisions have been made to incorporate compatibility with firmware V3.0. As the development progresses, the script will evolve to include additional features, tests, and enhancements, catering to a broader range of functionalities and accommodating various firmware versions.

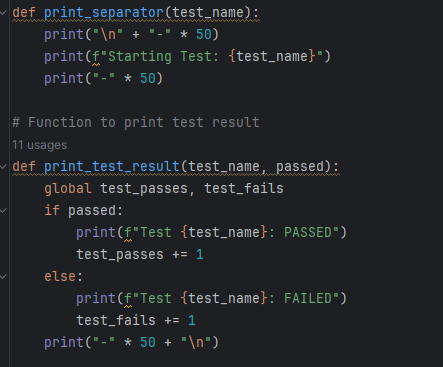
To run the script (Suggested IDE is PyCharm), user need to install Pythonnet (pip install pythonnet), import all libraries at the top of the code and add an interpreter to the project (Press add interpreter -> existing one -> your\_path\_\python1X.XX\python.exe).

**Libraries Used**

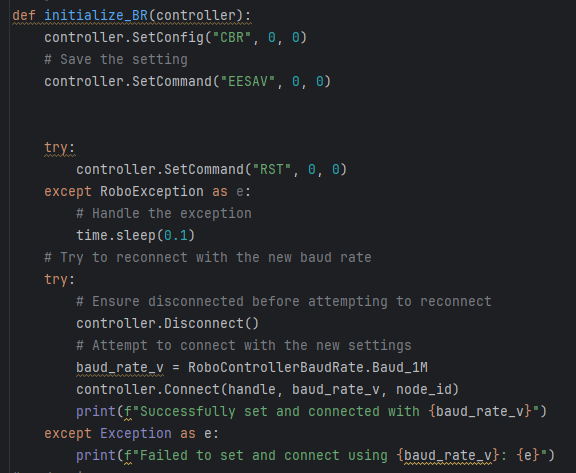
* clr: Allows accessing .NET libraries from Python.
* sys: Provides access to some variables used or maintained by the Python interpreter.
* time: Provides various time-related functions.
* re: Provides support for regular expressions.

**Functions**

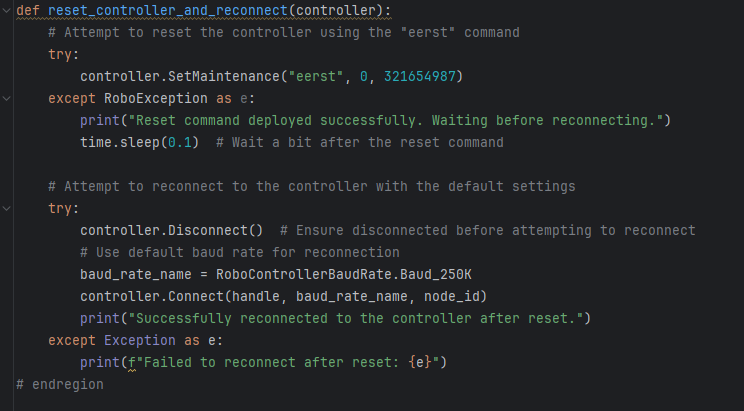
1. print\_separator(test\_name): Prints a separator indicating the start of a new test.
2. print\_test\_result(test\_name, passed): Prints the result of a test (passed or failed).



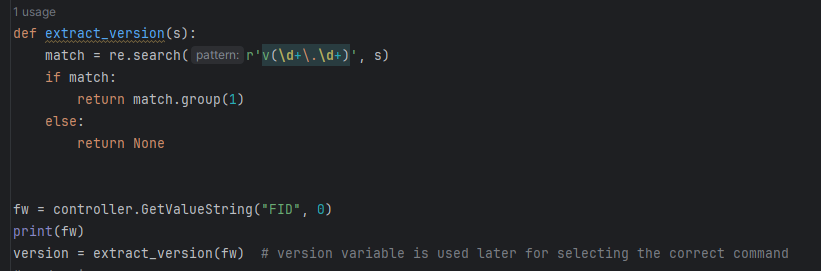
1. initialize\_BR(controller): Function to initialize the baud rate to 1M.



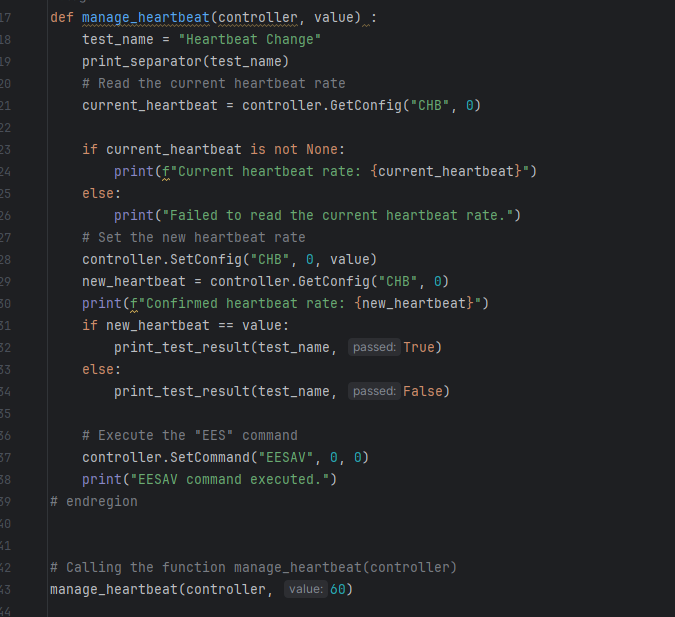
1. reset\_controller\_and\_reconnect(controller): Function to reset the controller to its default settings and reconnect.



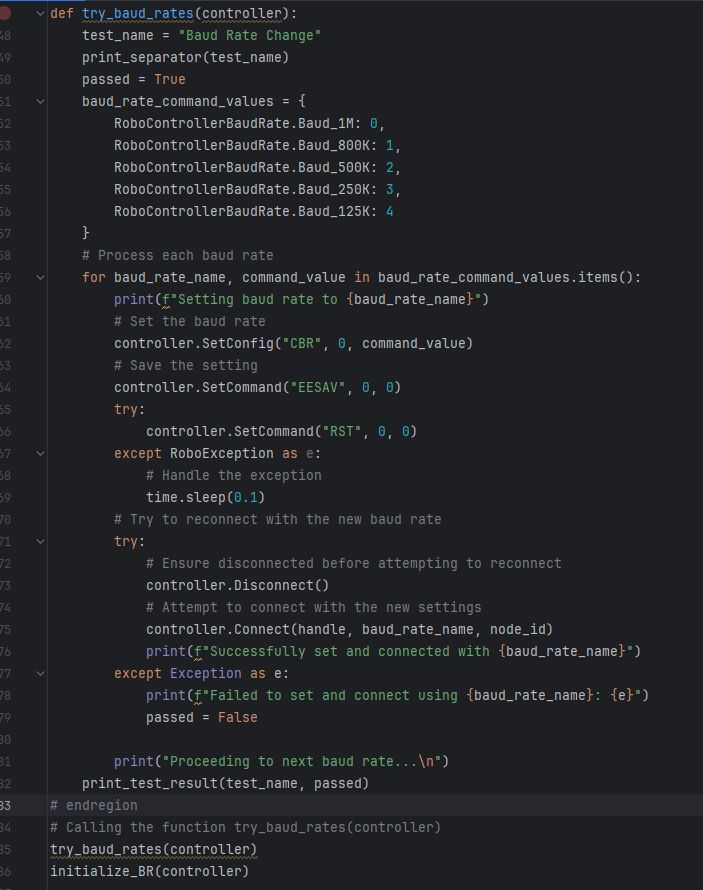
1. extract\_version(s): Helper function to extract the firmware version from a string in order to pick either V3.0 or V3.1 functions



1. manage\_heartbeat(controller, value): Performs a test to change the heartbeat rate.

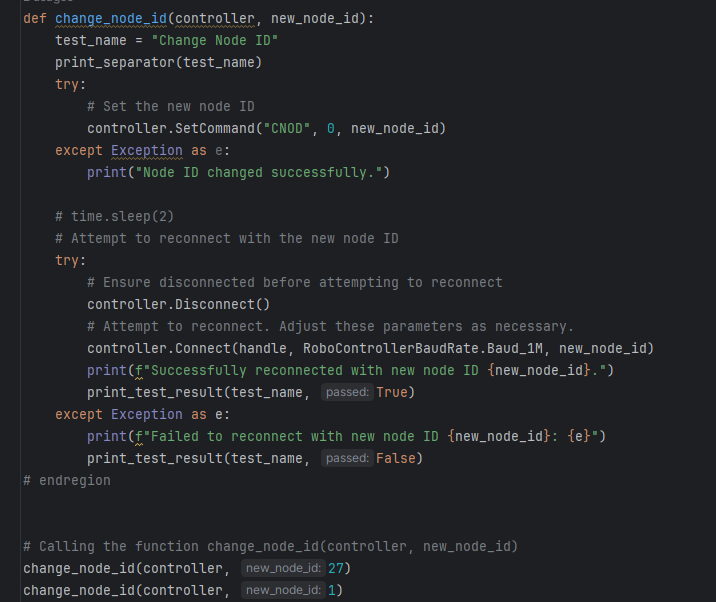


1. try\_baud\_rates(controller): Performs a test to change the baud rate to different values.

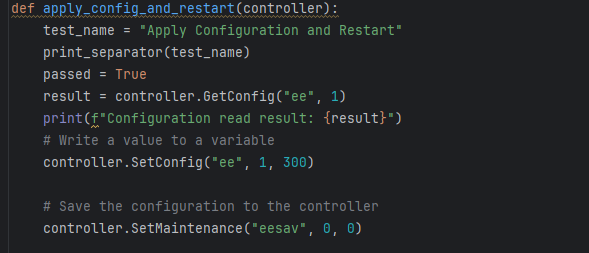


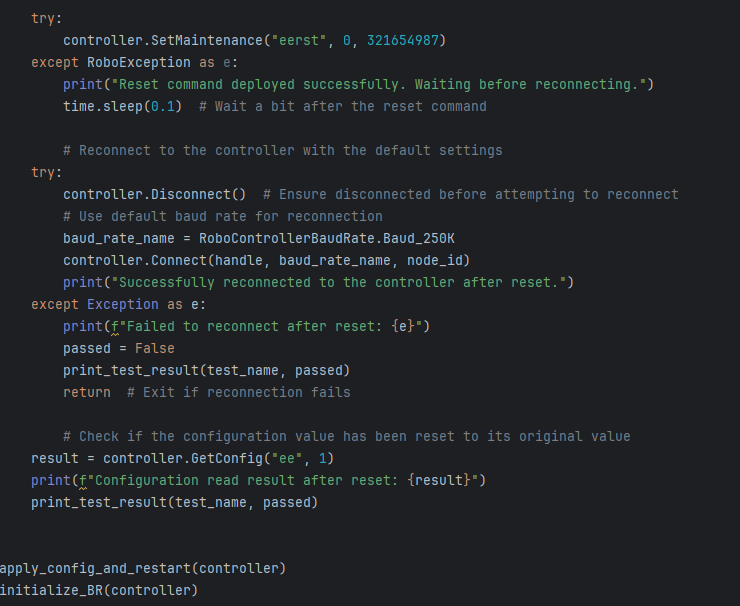
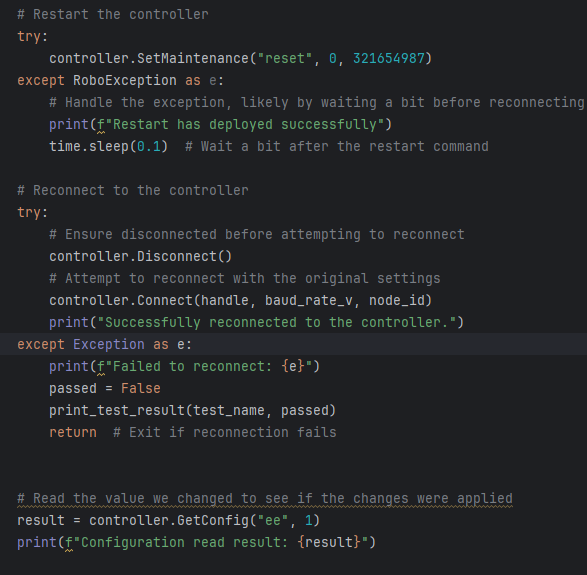
(After this function the initialize\_BR function is also called in order to set the baud rate back to 1M because the default value is 250k.

1. change\_node\_id(controller, new\_node\_id): Performs a test to change the node ID of the controller.

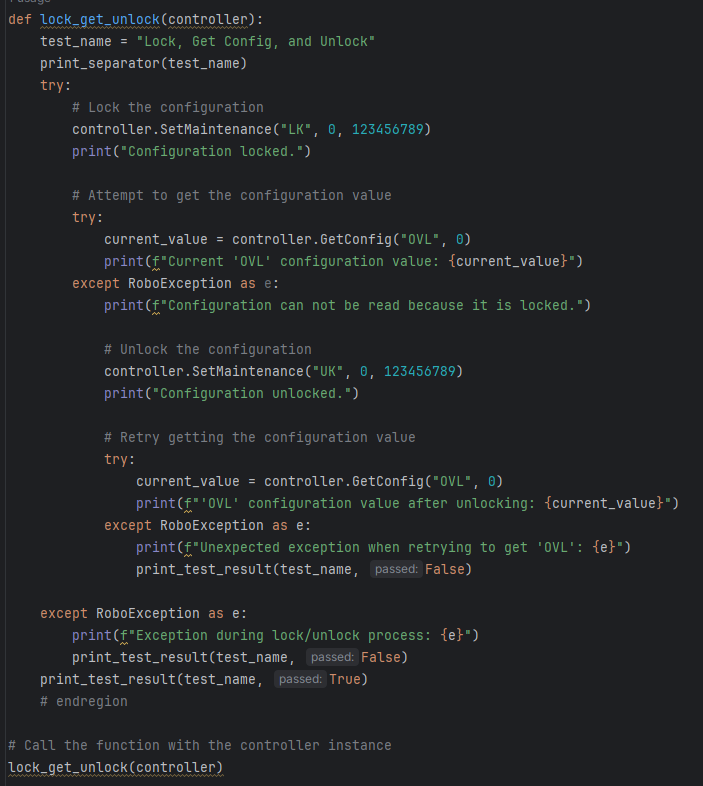


1. apply\_config\_and\_restart(controller): Applies configuration changes and restarts the controller.





1. lock\_get\_unlock(controller): Performs a test to lock, get configuration, and unlock.



**Script Execution**

1. Import necessary namespaces and initialize the Roboteq controller.
2. Find a valid adapter and connect to the controller.
3. Perform various tests and operations on the controller, including changing settings, restarting the controller, and checking its status.
4. Print the final report summarizing the number of passed and failed tests.

**Detailed Description**

* **Initializing the Controller**

1. The script starts by importing necessary namespaces and initializing the Roboteq controller using the provided DLL.
2. It finds a valid adapter and connects to the controller.
3. The script contains a function reset\_controller\_and\_reconnect(controller) to reset the controller to its default settings and reconnect.
4. This function resets the controller using the "eerst" command and attempts to reconnect to the controller with the default settings.

* **Changing Baud Rate**

1. The script contains a function try\_baud\_rates(controller) to test changing the baud rate to different values.
2. It iterates over different baud rates and attempts to set and connect to the controller with each baud rate.

* **Changing Node ID**

1. The script contains a function change\_node\_id(controller, new\_node\_id) to test changing the node ID of the controller.
2. It sets a new node ID and attempts to reconnect to the controller with the new node ID.

* **Applying Configuration Changes and Restarting**

1. The script contains a function apply\_config\_and\_restart(controller) to apply configuration changes to the controller and restart it.
2. It writes a value to a variable, saves the configuration, restarts the controller, and reconnects.
3. After this process has been completed successfully it resets the controller to default.

* **Locking and Unlocking Configuration**

1. The script contains a function lock\_get\_unlock(controller) to test locking, getting configurations and unlocking them.
2. It locks the configuration, attempts to get a configuration value, unlocks the configuration, and retries getting the configuration value.

* **Final Report**

1. After executing all tests, the script prints a final report summarizing the number of passed and failed tests.

**Conclusion**

This Python script provides a comprehensive set of tests and operations for interacting with a controller via CANOpen. It covers various aspects such as changing configuration, resetting the controller, and ensuring the functionality of the controller’s configuration and Maintenace commands.  
  
More tests were added included test for Fault Flags handling whose explanation is going to be documented soon.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
**FOR FW 3.0**

In firmware version 3.0, it's crucial to note that maintenance commands are not supported in CANOpen connections. Consequently, a significant portion of commands becomes inaccessible. This limitation extends to configuration commands, such as those responsible for adjusting the Baud Rate or Heartbeat rate, who are defined as READONLY commands in the .eds of the firmware. As a result, testing these functionalities effectively within the scope of this script becomes challenging. The logic of testing the V3.0 FW is by reading the .eds file. There you can find the sdo/pdo index of the commands and directly send the values you want.  
Explanation of the example of Heartbeat rate change for FW V3.0 :

**Function Definition:**

* The manage\_heartbeat function takes three parameters: controller, node\_id, and new\_heartbeat\_interval\_ms.
* controller is an instance of some controller class used for communication.
* node\_id represents the ID of the node (device) within the controller.
* new\_heartbeat\_interval\_ms is the new heartbeat interval specified in milliseconds.

**Setting SDO Parameters:**

* Two variables, heartbeat\_rate\_sdo\_index and heartbeat\_rate\_sdo\_sub\_index, are initialized with values representing the index and sub-index for the heartbeat rate Service Data Object (SDO)(found at .eds).
* These parameters are essential for accessing and modifying the heartbeat rate configuration on the controller.

**Reading Current Heartbeat Rate:**

* The current heartbeat rate is read from the controller using the ReadObjectS32 method of the RoboLabViewUtils.dll
* The method takes parameters such as timeout, node ID, SDO index, and sub-index to retrieve the current heartbeat rate.
* The retrieved heartbeat rate is stored in the current\_heartbeat\_rate variable.

**Setting New Heartbeat Rate:**

* The new\_heartbeat\_interval\_ms value is converted into a byte array in little-endian format.
* This byte array represents the new heartbeat interval to be written to the controller.
* The WriteObject method of the RoboLabViewUtils.dll is then used to write the new heartbeat interval to the controller.
* The method takes parameters like ReadObjectS32, including timeout, node ID, SDO index, sub-index, and data to write.
* The success of the write operation is stored in the success variable.

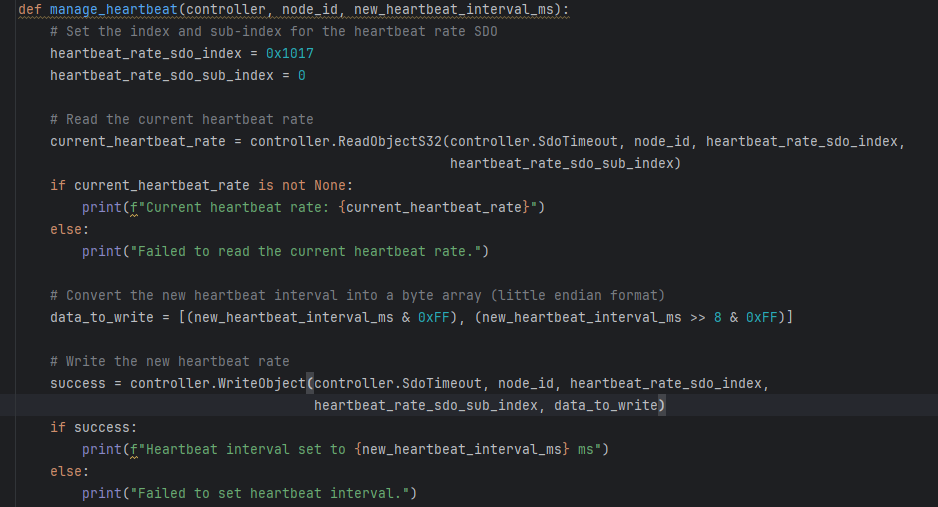
**Confirmation of New Heartbeat Rate:**

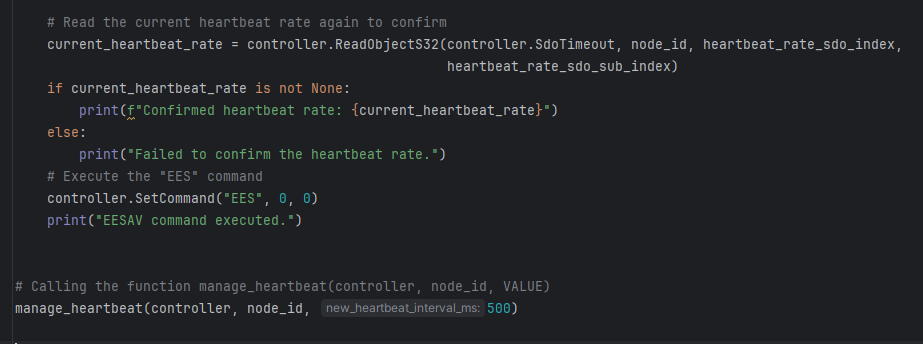
* After setting the new heartbeat rate, the script reads the current heartbeat rate again to confirm the change.
* This confirmation step ensures that the new heartbeat rate was successfully applied to the controller.

**Executing "EESAV" Command:**

1. The "EESAV" command is executed using the SetCommand method of the controller object.
2. This command typically saves the configuration changes made to the controller, ensuring they persist across power cycles or resets.
3. The command execution is printed to indicate its completion.

It is also worth noting that as mentioned before maintenance commands do not exist in FW V3.0. Although, EESAV and RESET commands have been implemented by turning them into runtime commands. Thus, they can be executed by sending SetCommand(“EES”, 0, 0) and SetCommand(“RST”, 0, 0).





**Expanded Documentation for Python Automation Script**

Referred to “restructured .Py”

**Overview**

This script automates testing of the RoboController through a series of systematic tests, which are designed to validate the controller's configuration, operational modes, and response to faults. This ensures that the controller is robust and behaves as expected under various scenarios.

**Prerequisites**

* Python 3.x
* pythonnet: Used to interact with the .NET DLL
* RoboLabViewUtils.dll: .NET library that needs to be in the specified directory

**Functions and Test Cases**

**Setup Functions**

“initialize\_BR(controller)”

**Purpose:** Resets the controller's baud rate to 1M and attempts to reconnect.

* Sets the baud rate configuration (CBR) to zero, effectively resetting it.
* Saves the setting using the EESAV command.
* Handles potential exceptions during the reset (RST) command, waiting 1 second if an exception occurs.
* Attempts to disconnect and reconnect with the new baud rate setting.

“reset\_controller\_and\_reconnect(controller)”

**Purpose:** Performs a soft reset of the controller and tries to reconnect using default settings.

* Sends a reset maintenance command (eerst) and waits briefly if it succeeds.
* Attempts to disconnect and reconnect using the default baud rate, handling any connection issues.

**Utility Functions**

“extract\_version(s)”

**Purpose:** Extracts the firmware version from a string, useful in logging or conditional operations based on firmware version.

* Uses regular expressions to find version patterns in a given string.
* “print\_separator(test\_name)” and “print\_test\_result(test\_name, passed)”

**Purpose:** Utility functions to format test output, making results clearer and more structured.

* “print\_separator” marks the start of a test.
* “print\_test\_result” logs whether a test passed or failed and updates global counters.

**Test Cases**

“manage\_heartbeat(controller, value)”

**Purpose:** Tests the controller's ability to change and confirm a heartbeat setting.

* Reads the current heartbeat configuration.
* Updates the heartbeat (CHB) and verifies the change.
* Results are validated and logged.

“try\_baud\_rates(controller)”

**Purpose:** Cycles through various baud rates to ensure the controller can operate reliably under each setting.

* Iteratively sets each baud rate from a predefined list and attempts to reconnect.
* Verifies connectivity at each baud rate, logging success or failure.

“change\_node\_id(controller, new\_node\_id)”

**Purpose:** Verifies the controller's ability to change its node ID and function with the new ID.

* Changes the node ID (CNOD) and checks for errors.
* Attempts to reconnect with the new node ID, verifying operational connectivity.

“apply\_config\_and\_restart(controller)”

**Purpose:** Tests configuration persistence through a controller reset.

* Sets a configuration value (ee), saves it, and then performs a controller reset.
* Verifies if the configuration persists post-reset by reconnecting and reading the value.

“lock\_get\_unlock(controller)”

**Purpose:** Tests the controller's configuration locking mechanism.

* Locks the configuration, verifies it is inaccessible, then unlocks and confirms access is restored.

**Fault Simulation Tests**

“test\_overvoltage\_protection(controller)” and “test\_undervoltage\_protection(controller)”

**Purpose:** Tests the controller's response to voltage faults.

* Manipulates overvoltage (OVL) and undervoltage (UVL) settings to trigger faults.
* Verifies that the controller correctly sets fault flags and clears them when conditions return to normal.

“test\_overtemp\_protection(controller)”

**Purpose:** Simulates an over-temperature condition to test the controller's thermal fault response.

* Sets the over-temperature limit (OTL) low to trigger a fault and then resets it, checking fault flag behavior.

“test\_estop\_protection(controller)”

**Purpose:** Tests the emergency stop functionality.

* Activates and then deactivates the emergency stop, verifying that the corresponding fault flags are set and cleared correctly.

“run\_motor\_and\_switch\_mode(controller)”

**Purpose:** Verifies motor control and mode switching functionality.

* Runs the motor in open-loop mode, stops it, switches to closed-loop speed mode, and runs again to confirm operational integrity in each mode.

**Mode Operation Tests**

“enable\_ds402\_and\_test\_modes(controller)”

**Purpose:** Enables DS402 standard functionality and cycles through various operational modes to verify each mode's proper activation.

* Enables DS402 via FSA.
* Cycles through a set list of modes, setting each one and verifying it with AOM.

**Execution**

The main() function initializes the controller, connects it, and sequentially executes each test, logging the overall results at the end. It's designed for extensibility and clarity, allowing easy additions and modifications.

**Final Report**

Summarizes the outcomes of all tests, providing a clear count of passed and failed tests. This is crucial for ongoing development and maintenance, ensuring that all functionalities are regularly verified.

--------------------------------------------------------------------------------------------------------------------------------------------------

This document should be used as a reference guide along with direct comments in the code for maximum clarity.