Introduction:

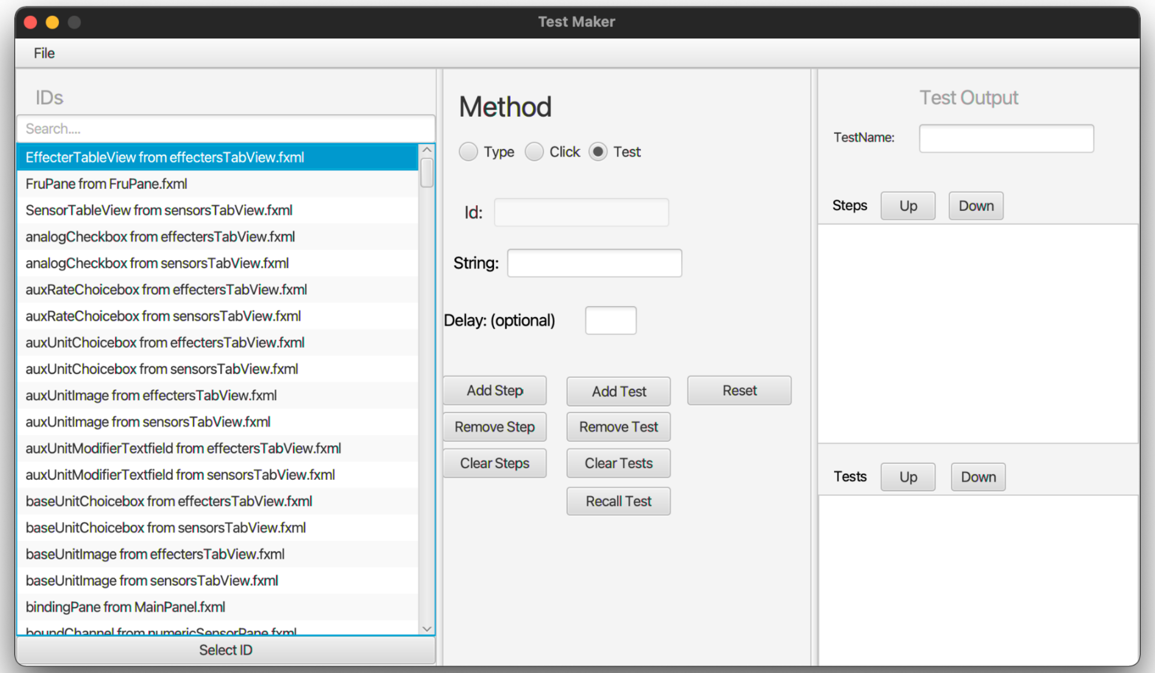
This is the documentation for the integration test maker. The goal of this application is to create tests that assert that the application is functioning correctly. In order to create these tests, the application creates json files that are converted to java classes. It should be noted that due to the limitations of the application, only one json file will be converted into a test at a time. Additionally, when another test is created it will override the current test. For the sake of organization, we will call the different json files configurations. How the flow of information works is that the user will add the different steps using the id tab and the method tab. Once a user is confident with their configuration they will save the steps into a test which can then be exported as a java file or saved as a json to work on later. If a user wishes to modify a configuration, they can load a previously saved configuration.

Running:

To run the integration test maker, right click on the Main.java file under iot_configurator/configurator/src/test/org.picmg.test/integrationTest/TestMaker and select "Run Test Maker".

Graphic Interface Components:

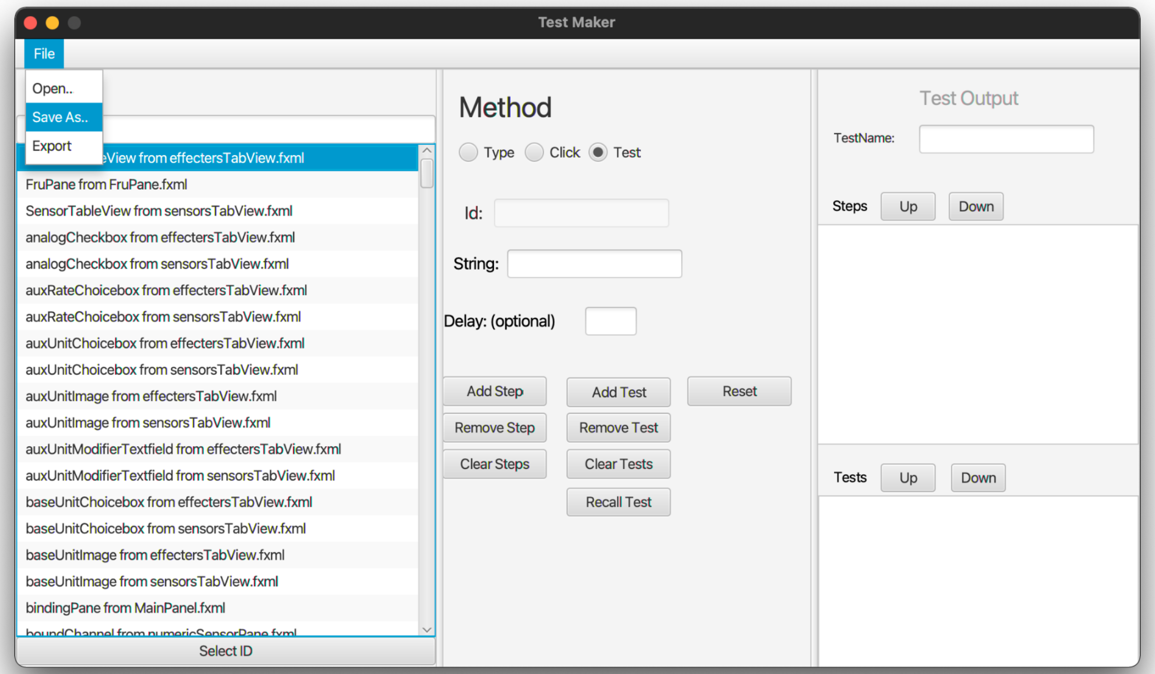
In this section we will go over the different components of the graphic user interface which will be referred to as the GUI. The Test Maker GUI is separated into 3 panes, the id pane, the method pane and the test output pane (Figure 1). Once a test is created, a user has 3 options located in the file menu; save as, load, and export. (Figure 2)



Type Click [✓ Test]

EffecterTableView from effectersTabView.fxml
FruPane from FruPane.fxml
SensorTableView from sensorsTabView.fxml
analogCheckbox from effectersTabView.fxml
analogCheckbox from sensorsTabView.fxml
auxRateChoicebox from effectersTabView.fxml
auxRateChoicebox from sensorsTabView.fxml
auxUnitChoicebox from effectersTabView.fxml
auxUnitChoicebox from sensorsTabView.fxml
auxUnitImage from effectersTabView.fxml
auxUnitImage from sensorsTabView.fxml
auxUnitModifierTextfield from effectersTabView.fxml
auxUnitModifierTextfield from sensorsTabView.fxml
baseUnitChoicebox from effectersTabView.fxml
baseUnitChoicebox from sensorsTabView.fxml
baseUnitImage from effectersTabView.fxml
baseUnitImage from sensorsTabView.fxml
bindingPane from MainPanel.fxml
boundChannel from numericSensorPane.fxml

Figure 1: TestMaker



Type Click [✓ Test]

View from effectersTabView.fxml
FruPane from FruPane.fxml
SensorTableView from sensorsTabView.fxml
analogCheckbox from effectersTabView.fxml
analogCheckbox from sensorsTabView.fxml
auxRateChoicebox from effectersTabView.fxml
auxRateChoicebox from sensorsTabView.fxml
auxUnitChoicebox from effectersTabView.fxml
auxUnitChoicebox from sensorsTabView.fxml
auxUnitImage from effectersTabView.fxml
auxUnitImage from sensorsTabView.fxml
auxUnitModifierTextfield from effectersTabView.fxml
auxUnitModifierTextfield from sensorsTabView.fxml
baseUnitChoicebox from effectersTabView.fxml
baseUnitChoicebox from sensorsTabView.fxml
baseUnitImage from effectersTabView.fxml
baseUnitImage from sensorsTabView.fxml
bindingPane from MainPanel.fxml
boundChannel from numericSensorPane.fxml

Figure 2: File Options in TextMaker

Id Pane:

The id pane is composed of three components; the search bar, the id list, and the id button. The search bar will look through all the ids available while ignoring the case of the input. The id list contains all the ids from all the fxml files that the configurator uses. Within each id entry, the file it comes from is also shown. Finally the id button puts the selected id into the id field location in the method pane.

Method Pane:

The method pane is composed of several components, thus we will break it down into three separate sections; radio buttons, text fields, and buttons.

Radio Button Section:

This section is made up of three radio buttons, each one representing a different action that can be called during the conversion of the configuration to a test. The first button is the type button, this button allows the user to type in a text field that they have previously clicked on. Only the string field is required to add this step to the output. The second button is the click button. The click button requires the id field to be filled in. Finally, the test button checks that the location, which uses the id field, is the same as what's in the string field.

Text Fields Section:

The text field section has three text fields in it. The first is the id field. This is where the different id's selected from the id pane will show up, additionally it is required for click and test radio buttons. The next button is the string field. The string field is the data the user wants to input into a specific location. It should be noted that it's assumed that a click step has occurred before a type step. Due to this assumption, the string field is required for the click step and the test step NOT the type step. The final field is the Delay field. This field is an optional field and without input will default to 1000. This delay is the delay in milliseconds that occurs between the different steps.

Buttons Section:

The buttons section is made of eight buttons, three buttons retaining to steps, four for testing, and one button for resetting.

Steps:

The three buttons that retain the steps are Add, Remove and Clear. The Add Step button will take the current method selected(Radio button) and the data from the Text Fields and create a Step in the test output pane. The next button is the Remove Step. The Remove Step button will remove the currently selected step in the test output pane from the steps list. Finally, the clear steps button will remove all the steps from the steps list located in the test output pane.

Test:

Similar to the test buttons, the test buttons have a matching button that does the exact same thing. The Add Test creates a new test by taking all the steps from the steps list and clearing it; the TestName field must be filled in to add a test. The Remove Test will remove a selected test from the test list in the test output pane. The Clear Test button will remove all the tests in the test list. The Final button is the Recall Test button. This button will take a selected test and recall the steps from it and add them to the step list.

General:

The one button in this section is the reset button. Once clicked, it will clear the Id, String and Delay fields.

Test output Pane:

Besides the TestName text field, this section is separated into 2 primary sections: the Steps list and Tests list. The added steps from the method pane are added to the steps list. From there, a user can move the step up or down using the up and down buttons. The same functionality is implemented for the test list.

Menu Options:

There are three menu options located in the File menu(See Figure 2). They are Open, Save as, and Export. Open will load a json file that has a configuration saved to it, this will bring in the test from it but not any steps that weren't saved to a test.

Test Reader:

In addition to the GUI functionality provided by the Export menu option, this functionality can be triggered through the command line. By running the TestReader.java file under iot_configurator/configurator/src/test/org.picmg.test/integrationTest/TestMaker in the command line, a test Java file can be generated. The file will be generated with the path to a valid JSON file representing the test. This JSON path should be passed as a command line argument when running the TestReader.java file. Provided a valid test JSON file was provided, the test file will be generated in the iot_configurator/configurator/src/test/org.picmg.test/integrationTest/generated folder.

TestMaker Command Documentation:

Commands can be run by adding a "/" character before certain messages in the type step of the TestMaker or serialized JSON.

The full list of commands is detailed in Figure 3.

Command	Description

/clear	Attempts to clear the focused item of the current window ● TextField/TextArea: clear text ● CheckBox: uncheck item ● ChoiceBox: select first item
/reset	Reload the root FXML component for the project.
/enter	Send an Enter keycode
/escape	Send an Escape keycode
/shift	Holds the shift key until /unshift command or program close
/unshift	Release the shift key
/ctrl	Holds the ctrl key until /unctrl command or program close
/unctrl	Release the ctrl key
/alt	Holds the alt key until /unalt command or program close
/unalt	Release the alt key
/key <keycode >	Passes any key given and translates it according to JavaFX keycode naming. https://openjfx.io/javadoc/16/javafx.graphics/javafx/scene/input/KeyCode.html#getKeyCode(java.lang.String)

Figure 3: TestMaker Commands

General Notes:

- If a command is passed with a space in it anywhere, it will not run the option unless the command takes a parameter (e.g. key).
- Commands are case insensitive.
- **It is recommended to let the program finish completely before exiting when using state-based key commands** (shift/unshift, alt/unalt, ctrl/unctrl). Commands that are not escaped at runtime, whether because a test forgets to escape a pressed key command or a runtime exception prevents termination, may cause some key presses to stick on the desktop after the test has quit. This effect may be reset by rerunning a test that successfully terminates (i.e. programmatically re-release the key), by restarting the effected pc, or by waiting for the OS to notice and resolve the issue (variable).

Walkthrough:

https://youtu.be/d1UNxW29hK4