# Train Module Test Plan Results

Shane Lester

### Model to Controller Communication

Inputs-

* Train Controller’s command
* Train’s inputs and limits

Outputs-

* Model’s acknowledgement of command
* Success or Failure of command notification

This test case shall be a test of communications between the Train Controller and Train Model work as planned. The model shall take a command issued by the train controller (be it opening the doors or giving it power etc.), and send back an acknowledgement that it received the command followed by notification of success or failure after action has been taken.

***Actual Result: Interactive-*** Train Model communications with Train Controller behave as planned.

***Tester:*** Shane Lester

***Date:*** 12/10/2012

***Notes:*** At first, my design had a class that would handle all of the communications between a train controller and model. After starting to code, I decided that just allowing the controller to create and manipulate everything it needed was simpler and more effective than the original plan.

***Acceptance:*** Acceptable

### Correct Movement Calculation

Inputs-

* Power
* Acceleration and Deceleration limits
* Authority

Outputs-

* Distance Moved
* Direction Moved
* Moved within Limits

This test case shall take the inputs that are needed from the track controller and confirm that the train model has moved the correct distance and in the specified direction. It shall also confirm that its movement was within the limits set by the controller.

***Actual Result: Interactive-*** Train gives back the distance it moved in one iteration of calculation. The train controller can either use a SetPointSpeed command in order to move the train, or can give it a power and then use the move method. Both of these methods correctly give the distance moved, and within the acceleration and deceleration limits. However, increasing speed using the SetPointSpeed command for small increases to speed takes a longer time than it should. For example: Going from 0-15 m/s takes 42 seconds, but going from 15-16 takes 38 more. This has been reported as an issue on GitHub.

***Tester:*** Shane Lester

***Date:*** 12/10/2012

***Notes:*** Potential problem in the Kp and Ki of the calculation being used in the SetPointSpeed method. Changing these variables may yield a better result.

***Acceptance:*** Functional but not fully correct.

### Station Behavior

Inputs-

* Door Control Commands
* Pre-Station number of passengers and total mass

Outputs-

* Door Control Status Checks
* Post-Doors Closed number of passengers and total mass

This test shall test the behavior of the train model when it is at a station. It shall check that the doors are opened and closed at the correct times, that the total number of passengers after the doors have been closed is correct, and that the total mass is calculated correctly with this number of passengers.

***Actual Result: Interactive-*** If stopped correctly at a stop block, the train model will correctly open and close the doors, and will correctly calculate the new number of passengers and update the total mass. The train controller shall be doing this manually. This test case works as was expected initially. However, there are some problems as of 12/11/2012 that it does not always stop at the right block for a stop on the track.

***Tester:*** Shane Lester

***Date:*** 12/11/2012

***Notes:*** It is not automated by the train model itself. The model has all the capabilities needed to do this, but the steps have to be done by the train controller.

***Acceptance:*** Acceptable

### Failure Detection and Action

Inputs-

* Simulated Failure (Engine Failure, Brake Failure, Signal Pickup failure, Emergency Break Thrown)

Outputs-

* Failure Detection Acknowledgement
* Correct Action taken according to the failure (or failures)

This test shall test the failure detection and action behavior of the train model. It shall take a simulated failure input by the tester, and shall test whether it is detected and the proper action is taken by the model.

***Actual Result: Interactive-*** Simulating failures and the action taken after these failures works correctly for the train model. If a failure is detected, the train is asked to stop, and does so accordingly. If the emergency brake is thrown, the deceleration limit is changed, and it is stopped. This way, stopping a train with the emergency brake takes less time than with regular deceleration limit.

***Tester:*** Shane Lester

***Date:*** 12/11/2012

***Notes:*** If the brakes fail, the emergency brake is thrown. This assumes that the normal brakes and the emergency brakes work independently.

***Acceptance:*** Acceptable

**Shane Lester**

# **Train Module Installation guide**

To be able to use the train model, one must simply install the Albion Train Control System package. All that is required in order to use the ATCS is a java virtual machine and java run time environment, which can both be downloaded and installed easily with instructions from oracle. After all files have been copied to the machine, it can be compiled and run as any other java program.

# **Train Module User Manual**

In the ATCS as a whole, the train model is only used for data and won’t show up anywhere in the main GUI. However, the train model itself does come with its own GUI.

In the top left section of the GUI is shown the limits on the train at the time. To the right of this, the total distance moved since beginning the UI is shown, and to the right of this the status of the doors can be seen. On the left side of the GUI, the attributes of the train can be seen, and they are updated as you interact with the GUI. To the right of this, the status of the critical components of the train model can be seen. On the rightmost part of the GUI is the log panel, which will show info when you move the train.

At the bottom at the buttons that are used to interact with the train itself. They are the following (in order from left to right, top down).

* **Simulate Passengers** – this button simulates the amount of passengers that get on / off the train at a stop. It will not go over the limit of passengers. It also updates the mass of the train.
* **Move** – this button uses the KeepMoving method of the train Model. It will keep the train moving at whatever speed it was currently going, but will not affect the force or acceleration. Results are shown in the log panel.
* **Set Point Speed** – this button will take whatever value you type into the section to the right of this button, and will input this into the SetPointSpeed method of the train model. Info on results is shown in the log panel.
* **Give power** – this button is used to manually give a certain power to the train, which is set to the field in the right. It is in kilowatts, it simply multiplies by 1000 for the GivePower method of the train model. Once the power has been given to the train, it uses the Move method to move the train one timestep. Info on results is shown in the log panel.
* **Toggle Doors** – simply toggles the doors open or closed
* **Sim Brake Failure** – simulates a brake failure and updates the log and failure panels.
* **Sim Signal Failure** – simulates a signal failure and updates the log and failure panels.
* **Sim Engine Failure** – simulates an engine failure and updates the log and failure panels.
* **Throw emergency brake** – simulates the effects of someone throwing the emergency brake.
* **Exit**- exits the GUI.

# **Train Module Configuration Management**

For version control and configuration management of future iterations of the train model, top level functionality and the way it is used by other modules must stay the same. In future versions of the model, better movement calculations would be implemented, but it must also be compatible with older versions of the train controller. If a big change must be done that would require an update of the train controller, it must be documented and added as an update with the train controller as well. In that case it cannot be done by itself, or it would cause other modules to fail.