Train Trax: Train Monitor for Positive Train Control Test Beds

Software Test Description

Version 1.0

Mm/dd/yy

Stephen Jalbert

Rashad Madyun

Corey Sanders

|  |  |  |  |
| --- | --- | --- | --- |
| Revision History | | | |
| Version | Date | Description | Author |
| 1.0 | 03/18/2016 | Initial Version. Created template for Software Test Description. Included content for testing MDU and the Train Navigation Database. | Stephen Jalbert  Rashad Madyun  Corey Sanders |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Revision History 2](#_Toc448857720)

[1 Introduction 5](#_Toc448857721)

[1.1 Purpose of this document 5](#_Toc448857722)

[1.2 System Overview 5](#_Toc448857723)

[1.3 Document Overview 5](#_Toc448857724)

[2 Referenced Documents 6](#_Toc448857725)

[3 Test Preparations 6](#_Toc448857726)

[4 Test Descriptions 6](#_Toc448857727)

[4.1 Motion Detection Unit Tests 6](#_Toc448857728)

[4.1.1 Purpose 6](#_Toc448857729)

[4.1.2 Description 6](#_Toc448857730)

[4.1.3 Required Tools 6](#_Toc448857731)

[4.1.4 General Test Procedures 7](#_Toc448857732)

[4.1.5 Test Cases 9](#_Toc448857733)

[4.2 Train Navigation Database Tests 9](#_Toc448857734)

[4.2.1 Purpose 9](#_Toc448857735)

[4.2.2 Description 10](#_Toc448857736)

[4.2.3 Required Tools 10](#_Toc448857737)

[4.2.4 General Test Procedures 10](#_Toc448857738)

[4.2.5 Test Cases 13](#_Toc448857739)

[4.3 Train Navigation Service Tests 15](#_Toc448857740)

[4.3.1 Purpose 15](#_Toc448857741)

[4.3.2 Description 15](#_Toc448857742)

[4.3.3 Required Tools 15](#_Toc448857743)

[4.3.4 General Test Procedures 16](#_Toc448857744)

[4.3.5 Test Cases 19](#_Toc448857745)

[4.4 Train Navigation GUI Tests 29](#_Toc448857746)

[4.4.1 Purpose 29](#_Toc448857747)

[4.4.2 Description 29](#_Toc448857748)

[4.4.3 Required Tools 29](#_Toc448857749)

[4.4.4 Android Train Trax GUI Test Procedures 29](#_Toc448857750)

[4.4.5 Test Cases 31](#_Toc448857751)

[5 Requirements Traceability 35](#_Toc448857752)

[5.1 Traceability from Test Case to requirement 35](#_Toc448857753)

[5.1.1 Train Navigation Database 35](#_Toc448857754)

[5.1.2 [TBD] Table of Test Driver tests here 46](#_Toc448857755)

[5.2 Traceability Matrices 46](#_Toc448857756)

[5.2.1 Train Navigation Database 46](#_Toc448857757)

[6 Notes 47](#_Toc448857758)

[7 Appendixes 47](#_Toc448857759)

# Introduction

## Purpose of this document

The purpose of this document is to describe the tests performed to verify the design og the Train Trax Train Monitor system. It will detail each test and the hardware and software configuration necessary for each test.

## System Overview

Train Trax's primary purpose is to estimate the position of each train operating along the Positive Train Control Test Bed accurately enough to allow Train Operators schedule trains to run close enough to operation on the same section of track with minimal risk of collision. Additionally, Train Trax provides a means for Train Operators to easily control switches on the train track without the need to using any additional train control software. Train Trax is only a monitor for trains, not train control software. Furthermore, the development team is to assist the department with any modifications necessary to the Positive Train Control Test Bed to support proper operation of Train Trax, including the placement of markers on the track at pre-designated locations.

Train Trax consists of hardware that is equipped on either the train engine or rail cars to measure train movement. It also consists of software that will run on existing equipment within the department to graphically display train positions and to control movement.

A unit is attached to a rail car that is equipped with an Inertial Motion Unit (IMU) that measures the acceleration and angular velocity (rotational vectors) of the rail car as it is tugged by the train along the track. This unit, called a Motion Detection Unit, will send its collected measurements over WIFI to a train monitor terminal (i.e. computer) that will estimate the train’s position using numerical integration to solve for displacement kinematic equations. The resulting position is then displayed on the terminal as well as the layout of the track itself. RFID tags, whose position is already recorded in a database, will be used as the track markers and placed strategically throughout the track so that they can correct the position calculated from IMU measurements. Lastly, the monitor terminal displays representations of all of the switches on the track and allows the user to control them through a GUI that sends LOCONET messages to the track's switch controllers, which then control relays to change a switch’s state. Train control software, such as JMRI, is expected to be used to control/throttle the movement of the train via LOCONET messages to the Train Command Station.

## Document Overview

The remainder of the STD will provide a detailed description of each test used to prove system functionality.

# Referenced Documents

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

# Test Preparations

The preparations necessary for each test will be described in each test description.

# Test Descriptions

## Motion Detection Unit Tests

### Purpose

The purpose of this section to describe the series of tests necessary to verify that all of the requirements for the Motion Detection Unit of Train Trax are met. The Motion Detection Unit consists of hardware and software that runs on each train to sense and report train motion and RFID tag detection to the Train Navigation Database for use by other Train Trax subsystems.

### Description

Testing is conducted in accordance to the Train Trax Test Plan. Testing for this subsystem is primarily is conducted by using the MDU Utility Test Driver console program operated by the Tester. Test Results from the Test Driver should be recorded as the primary artifact for verification that testing requirements work correctly. The subsections that follow provide detailed information on how to conduct testing for the Motion Detection Unit.

### Required Tools

* Arduino 1.6.6
* MDU utility test driver
* USB cord and FIDI Basic
* MDU Hardware

### General Test Procedures

#### Using the Test Case/Test Procedure Tables

For the remainder of this testing document, it is expected that the Test Operator do the following:

1. Perform the actions described in the Action Column of each Test Case/Test Procedure table.
2. Determine if the behavior / output described in the ‘Expected Result’ Column happens after executing the corresponding action.
3. If the ‘Expected Result’ is observed, the fill the ‘Pass/Fail’ column with Pass. Otherwise, fill the ‘Pass/Fail’ column with Fail.

#### MDU Configuration

Test sequence describes what is necessary to open, configure, and program the source code for the MDU.

|  |  |  |
| --- | --- | --- |
| Description | Expected Result | Pass/Fail |
| 1. Open Arduino 1.6.6 | The Arduino application should start and appear on screen. |  |
| 1. Open MDU Source Code | Select the File menu and select open. Navigate to the CPE656TL/source/MDUsource/MDU\_2.0 directory. Select and open MDU\_2.0.ino. |  |
| 1. Configure MDU Source | At the top of the Arduino application, select the config.h tab. Modify the TRAIN\_ID value to the hex value of the MDU hardware’s ID. (e.g. Train 26 should have TRAIN\_ID = 0x1a) |  |
| 1. Connect the MDU hardware to programming computer | Disconnect the XBee from the horizontal side pins on the Arduino. Connect that FIDI Basic such that the RXI pin connects to the TXO pin on the Arduino. Attach the USB cable to the FIDI Basic and to a USB port on the programming computer. |  |
| 1. Configure Project for programming | Select the Tools Tab and verify that the Board field lists: “Arduino Pro or Pro Mini” if not select that option under the expansion window. Under the same tab, verify that the Processor field lists: “Atmega328 (3.3V, 8MHz)” if not select it under the expansion window. Record the value of the Port Field as it will be necessary to configure the MDU utility test driver. |  |
| 1. Prepare the MDU Hardware for programming | Locate the RXI pin on the vertical headers. This pin connects the RFID tag reader to the Arduino, and must be removed to program the MDU. Disconnect this wire. |  |
| 1. Program the MDU | Select the arrow icon in the top left of the Arduino to begin compiling and programming. |  |
| 1. Prepare the MDU for use. | When the Arduino application completes programming, as indicated on the status bar in the lower left of the program, reconnect the RFID tag reader pin to the vertical RXI pin of the Arduino. Disconnect the FIDI basic and connect the XBee cable to the horizontal RXI and TXO pins as indicated on the connector. Connect the 9v battery to the battery connector. Connect the computer side XBee to the USB port the FIDI basic was connected to. |  |

#### Running the Test Driver

A description is provided on the use of the MDU utility test driver for testing the function of the MDU.

|  |  |  |
| --- | --- | --- |
| Description | Expected Result | Pass/Fail |
| 1. Open the MDU utility test driver. | Navigate to the CPE656TL\source\MDUutilities\MDUutd directory and select the MDUutd.exe and run the program. A console program should launch. |  |
| 1. Configure the MDU utility test driver | The program should display a prompt for a Serial Port. Input the port number that was recorded from the Arduino application. A connection verification message should appear. |  |

### Test Cases

#### MDU Tests

##### Sensor Outputs Test Case

###### Description

The test verifies that the MDU can report sensor data correctly to the Serial Data Link.

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘MDU Configuration’ test procedure. | The MDU is properly programed and configured and is connected to the XBee data link. |  |
| 1. Perform the steps described in the ‘Running the Test Driver’ test procedure. | The MDU utilities test driver is connected to the Serial connection with the MDU. |  |
| 1. The MDU utilities test driver will prompt for display mode selection. Select view IMU outputs. Close the MDU utilities application upon verification. | The MDU utilities test driver should continuously output received IMU data in the form: Train ID, Time Stamp, Accelerometer(x,y,z), Gyroscope(x,y,z). |  |
| 1. Perform the steps described in the ‘Running the Test Driver’ test procedure. | The MDU utilities test driver is connected to the Serial connection with the MDU. |  |
| 1. The MDU utilities test driver will prompt for display mode selection. Select view RFID outputs. Move the train across track RFID tags and verify that the tags are reported on the MDU utilities application. Close the MDU utilities application upon verification. | The MDU utilities test driver should output a RFID received message after the train passes over each track mounted RFID tag that contains a time stamp and the RFID tag hex ID. |  |

## Train Navigation Database Tests

### Purpose

The purpose of this section to describe the series of tests necessary to verify that all of the requirements for the Train Navigation Database subsystem of Train Trax are met. The Train Navigation Database is library that is used by other Train Trax subsystems to persist information necessary for system operation and improvement. It is primarily responsible for retaining information about the shape and size of the Positive Train Control Test Bed (Track Geometry). This is necessary so that Train Trax can correlate train movement with its position along the Test Bed tracks. It is also responsible for saving position estimates and raw measurements used to derive position estimates.

### Description

Testing is conducted in accordance to the Train Trax Test Plan. Testing for this subsystem is primarily conducted through a series of integration tests created to run from the Junit unit testing framework. However, some tests are also conducted by a TestDriver console program that guides the Tester through a series of steps for testing operations that require a human to be in the loop.

Test Results from the Test Driver and the automated integration tests should be recorded as the primary artifact for verification that testing requirements work correctly. The Junit plugin from Eclipse is issued to launch the tests directly from the source projects.

The subsections that follow provide detailed information on how to conduct testing for the Train Navigation Database subsystem.

### Required Tools

* JUnit 4
* Eclipse Mars (4.5)
* My SQL Server (5.5.47)
* Train Trax My SQL Table Creation Script
* Java 7
* MYSQL JDBC Driver (Connector/J 5.1.38)

### General Test Procedures

#### Using the Test Case/Test Procedure Tables

For the remainder of this testing document, it is expected that the Test Operator do the following:

1. Perform the actions described in the Action Column of each Test Case/Test Procedure table.
2. Determine if the behavior / output described in the ‘Expected Result’ Column happens after executing the corresponding action.
3. If the ‘Expected Result’ is observed, the fill the ‘Pass/Fail’ column with Pass. Otherwise, fill the ‘Pass/Fail’ column with Fail.

#### Open A Train Trax Workspace

Test sequence describes what is necessary to get the eclipse configured to run testing for the Train Navigation Database.

|  |  |  |
| --- | --- | --- |
| Description | Expected Result | Pass/Fail |
| 1. Open Eclipse | The eclipse application should appear upon the screen. |  |
| 1. Examine the Project Explorer window. | All of the Train Trax Projects in the CPE656TL/source directory should appear in the Project Explorer in the left hand corner of the table.  No further steps are necessary if projects are present.  If they are not present, then proceed to the next step. |  |
| 1. Click on File->Import. | The Import window should appear. |  |
| 1. Click on ‘Existing Projects into Workspace’, then click Next. | The Import window should update to the ‘Import Projects’ screen. |  |
| 1. Click on the Browse button to the right of the ‘Select root directory’ text box. | A Browse window should appear. |  |
| 1. Browse to the location of the CPE656TL/source directory. Click OK. | A list of Train Trax Projects should appear in the Projects List Box. |  |
| 1. Click on the ‘Select All’ button. | All of the Train Trax Projects should have a check mark beside their listing in the Projects List Box. |  |
| 1. Click on the Finish Button. | All of the Train Trax Projects in the CPE656TL/source directory should appear in the Project Explorer in the left hand corner of the table. |  |

#### Running Integration Tests

A description is given on how to use Eclipse to run all of the integration tests defined for the Train Navigation Database.

|  |  |  |
| --- | --- | --- |
| Description | Expected Result | Pass/Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test sequence. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests project. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the results from the Junit tab. | All of the tests are run. This is indicated by the following ratio: (Tests Run / Tests Available). For example, if there are 83 tests then, it should report in the Runs field “83/83”.  The Errors field reports “0”.  The Failures field reports “0”. |  |

#### Running the Test Driver

A description is given on how to use Eclipse to run the Train Navigation Database Test Driver to complete human-in the-loop testing.

|  |  |  |
| --- | --- | --- |
| Description | Expected Result | Pass/Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test sequence. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.TestDriver project. | A context menu appears. |  |
| 1. Click on Run As -> Java Application | A console window appears. |  |
| 1. Follow the steps described by the console window. | A sequence of tests is conducted where the program prompts the operator for input at stages.  When testing is complete, the program should return the following: A prompt reporting “Testing Complete” in the console window, the file path for the test results file, and the final outcome of the test driver testing: “Pass” or “Fail”. |  |

### Test Cases

#### TrackBlockRepository Tests

##### TestFindWithAllSearchCriteria Test Case

###### Description

The test verifies that track blocks can be found when searches against all of the values of the columns in the track block table are performed.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackBlockRepository class instance. | Junit creates a TrackBlockRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Block class instance with the following values:   BlockName: “Name” | A track block entry instance is created with the specified values. |
| 1. Add the Track Block class instance to the Track Block Repository by calling TrackBlockRepository.Add | The track block entry is added to the database. The TrackBlockRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > * BlockName: “Name”   An exception is not raised by the test case. |
| 1. Search the TrackBlockRepository by calling TrackBlockRepository.Find with the following Search Criteria:   BlockName: “Name” | The TrackBlockRepository Find method returns exactly one match. The match has the following information:   * Id: < Automatically Assigned ID from the Database > * BlockName: “Name”   An exception is not raised by the test case. |
| 1. Remove the added entry using TrackBlockRepository.Remove | The TrackBlockRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackBlockRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestFindWithAllSearchCriteria test results | The TestFindWithAllSearchCriteria test should have a green check mark and be reported as ‘Success’. |  |

##### TestFindById Test Case

###### Description

The test verifies that track blocks can be found when searching by the primary key (or ID) associated with the target entry in the track block table.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackBlockRepository class instance. | Junit creates a TrackBlockRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Block class instance with the following values:   BlockName: “Name” | A track block entry instance is created with the specified values. |
| 1. Add the Track Block class instance to the Track Block Repository by calling TrackBlockRepository.Add | The track block entry is added to the database. The TrackBlockRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > (Referred to as ID1) * BlockName: “Name”   An exception is not raised by the test case. |
| 1. Search the TrackBlockRepository by calling TrackBlockRepository.Find with the following Parameters:   Id: <ID1> | The TrackBlockRepository Find method returns exactly one match. The match has the following information:   * Id: <ID1> * BlockName: “Name”   An exception is not raised by the test case. |
| 1. Remove the added entry using TrackBlockRepository.Remove | The TrackBlockRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackBlockRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestFindById test results | The TestFindById test should have a green check mark and be reported as ‘Success’. |  |

##### TestAdd Test Case

###### Description

The test verifies that new track block entries are correctly added to the track block table.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackBlockRepository class instance. | Junit creates a TrackBlockRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Block class instance with the following values:   BlockName: “Name” | A track block entry instance is created with the specified values. |
| 1. Add the Track Block class instance to the Track Block Repository by calling TrackBlockRepository.Add | The track block entry is added to the database. The TrackBlockRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > * BlockName: “Name”   An exception is not raised by the test case. |
| 1. Compare the track block value returned by the repository from the add operation against the track block value that was added. | The two values match. The match has the following information:   * Id: < Automatically Assigned ID from the Database > * BlockName: “Test”   An exception is not raised by the test case. |
| 1. Remove the added entry using TrackBlockRepository.Remove | The TrackBlockRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackBlockRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestAdd test results | The TestAdd test should have a green check mark and be reported as ‘Success’. |  |

##### TestUpdate Test Case

###### Description

The test verifies that track block entries are correctly updated to the track block table.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackBlockRepository class instance. | Junit creates a TrackBlockRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Block class instance (referred to as Instance A) with the following values:   BlockName: “Name” | A track block entry instance is created with the specified values. |
| 1. Add the Track Block class instance to the Track Block Repository by calling TrackBlockRepository.Add | The track block entry is added to the database. The TrackBlockRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: <Automatically Assigned ID from the Database> (Referred to in here as ID1) * BlockName: “Name”   An exception is not raised by the test case. |
| 1. Change the following information for instance A using the corresponding set methods:   BlockName: “Name1” | Instance A now has the following values:  BlockName: “Name1” |
| 1. Update instance A’s repository entry in the TrackBlock repository by calling TrackBlockRepository.Update | Instance A’s track block entry is updated in the database. The TrackBlockRepository.Update method returns a RepositoryEntry instance with the following value:   * Id: <ID1> * BlockName: “Name1”   An exception is not raised by the test case. |
| 1. Search for instance A by ID using TrackBlockRepository.FindById:   ID: <ID1> | Instance A’s track block entry in the database is found with the following information:   * Id: <ID1> * BlockName: “Name1”   An exception is not raised by the test case. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackBlockRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestUpdate test results | The TestUpdate test should have a green check mark and be reported as ‘Success’. |  |

##### TestRemove Test Case

###### Description

The test verifies that track block entries are correctly removed from the track block table. A track block entry is added, then removed, and lastly testing is done to verify that it was correctly removed by searching for the deleted item.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackBlockRepository class instance. | Junit creates a TrackBlockRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Block class instance (referred to as Instance A) with the following values:   BlockName: “Name” | A track block entry instance is created with the specified values. |
| 1. Add the Track Block class instance to the Track Block Repository by calling TrackBlockRepository.Add | The track block entry is added to the database. The TrackBlockRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > (Referred to as ID1) * BlockName: “Name”   An exception is not raised by the test case. |
| 1. Remove the Track Block instance A from the TrackBlockRepository using the it’s repository entry ID. | The row in the TrackBlock Database is removed. (i.e. The track block entry in the repository is removed). |
| 1. Search for the deleted repository entry using the repository entry ID: ID1. | No matches are found in the repository (i.e. database) in the search.  An exception is not raised by the test case. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackBlockRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestRemove test results | The TestRemove test should have a green check mark and be reported as ‘Success’. |  |

#### TrackPointRepository Tests

##### TestFindWithAllSearchCriteria Test Case

###### Description

The test verifies that track points can be found when searches against all of the values of the columns in the track point table are performed.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackPointRepository class instance. | Junit creates a TrackPointRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Point class instance with the following values:   PointName: “Name” | A track point entry instance is created with the specified values. |
| 1. Add the Track Point class instance to the Track Point Repository by calling TrackPointRepository.Add | The track point entry is added to the database. The TrackPointRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > * PointName: “Name”   An exception is not raised by the test case. |
| 1. Search the TrackPointRepository by calling TrackPointRepository.Find with the following Search Criteria:   PointName: “Name” | The TrackPointRepository Find method returns exactly one match. The match has the following information:   * Id: < Automatically Assigned ID from the Database > * PointName: “Name”   An exception is not raised by the test case. |
| 1. Remove the added entry using TrackPointRepository.Remove | The TrackPointRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestFindWithAllSearchCriteria test results | The TestFindWithAllSearchCriteria test should have a green check mark and be reported as ‘Success’. |  |

##### TestFindById Test Case

###### Description

The test verifies that track points can be found when searching by the primary key (or ID) associated with the target entry in the track point table.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackPointRepository class instance. | Junit creates a TrackPointRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Point class instance with the following values:   PointName: “Name” | A track point entry instance is created with the specified values. |
| 1. Add the Track Point class instance to the Track Point Repository by calling TrackPointRepository.Add | The track point entry is added to the database. The TrackPointRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > (Referred to as ID1) * PointName: “Name”   An exception is not raised by the test case. |
| 1. Search the TrackPointRepository by calling TrackPointRepository.Find with the following Parameters:   Id: <ID1> | The TrackPointRepository Find method returns exactly one match. The match has the following information:   * Id: <ID1> * PointName: “Name”   An exception is not raised by the test case. |
| 1. Remove the added entry using TrackPointRepository.Remove | The TrackPointRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests -> TrackPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestFindById test results | The TestFindById test should have a green check mark and be reported as ‘Success’. |  |

##### TestAdd Test Case

###### Description

The test verifies that new track point entries are correctly added to the track point table.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackPointRepository class instance. | Junit creates a TrackPointRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Point class instance with the following values:   PointName: “Name” | A track point entry instance is created with the specified values. |
| 1. Add the Track Point class instance to the Track Point Repository by calling TrackPointRepository .Add | The track point entry is added to the database. The TrackPointRepository .Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > * PointName: “Name”   An exception is not raised by the test case. |
| 1. Compare the track point value returned by the repository from the add operation against the track point value that was added. | The two values match. The match has the following information:   * Id: < Automatically Assigned ID from the Database > * PointName: “Test”   An exception is not raised by the test case. |
| 1. Remove the added entry using TrackPointRepository.Remove | The TrackPointRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestAdd test results | The TestAdd test should have a green check mark and be reported as ‘Success’. |  |

##### TestUpdate Test Case

###### Description

The test verifies that track point entries are correctly updated to the track point table.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackPointRepository class instance. | Junit creates a TrackPointRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Block class instance (referred to as Instance A) with the following values:  * PointName: “Name” * Type: “Point” * X: 1 * Y: 1 * Z: 1 * Blockid: “2” * TagName: “Tag” | A track point entry instance is created with the specified values. |
| 1. Add the Track Point class instance to the Track Point Repository by calling TrackPointRepository .Add | The track point entry is added to the database. The TrackPointRepository .Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > (Referred to as ID1) * PointName: “Name” * Type: “Point” * X: 1 * Y: 1 * Z: 1 * Blockid: “2” * TagName: “Tag”   An exception is not raised by the test case. |
| 1. Change the following information for instance A using the corresponding set methods:   PointName: “Name1” | Instance A now has the following values:   * PointName: “Name1” |
| 1. Update instance A’s repository entry in the TrackPoint repository by calling TrackPointRepository.Update | Instance A’s track point entry is updated in the database. The TrackPointRepository.Update method returns a RepositoryEntry instance with the following value:   * Id: <ID1> * PointName: “Name1” * Type: “Point” * X: 1 * Y: 1 * Z: 1 * Blockid: “2” * TagName: “Tag”   An exception is not raised by the test case. |
| 1. Search for instance A by ID using TrackPointRepository.FindById:   ID: <ID1> | Instance A’s track point entry in the database is found with the following information:   * Id: <ID1> * PointName: “Name1” * Type: “Point” * X: 1 * Y: 1 * Z: 1 * Blockid: “2” * TagName: “Tag”   An exception is not raised by the test case. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestUpdate test results | The TestUpdate test should have a green check mark and be reported as ‘Success’. |  |

##### TestRemove Test Case

###### Description

The test verifies that track point entries are correctly removed from the track point table. A track point entry is added, then removed, and lastly testing is done to verify that it was correctly removed by searching for the deleted item.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a TrackPointRepository class instance. | Junit creates a TrackPointRepository instance. An exception is not raised by the test case. |
| 1. Create a Track Point class instance (referred to as Instance A) with the following values:   PointName: “Name” | A track point entry instance is created with the specified values. |
| 1. Add the Track Point class instance to the Track Point Repository by calling TrackPointRepository .Add | The track point entry is added to the database. The TrackPointRepository .Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > (Referred to as ID1) * PointName: “Name”   An exception is not raised by the test case. |
| 1. Remove the Track Point instance A from the TrackPointRepository using the it’s repository entry ID. | The row in the TrackPoint Database is removed. (i.e. The track point entry in the repository is removed). |
| 1. Search for the deleted repository entry using the repository entry ID: ID1. | No matches are found in the repository (i.e. database) in the search.  An exception is not raised by the test case. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->TrackPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestRemove test results | The TestRemove test should have a green check mark and be reported as ‘Success’. |  |

#### AdjacentPointRepository Tests

##### TestFindWithAllSearchCriteria Test Case

###### Description

The test verifies that adjacent track points can be found when searches against all of the values of the columns in the adjacent track point table are performed.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a AdjacentPointRepository class instance. | Junit creates a AdjacentPointRepository instance. An exception is not raised by the test case. |
| 1. Create a AdjacentPoint class instance with the following values:  * PointId: 1 * AdjacentPointId: 2 | An adjacent track point entry instance is created with the specified values. |
| 1. Add the AdjacentPoint class instance to the AdjacentPoint Repository by calling AdjacentPointRepository.Add | The adjacent track point entry is added to the database. The AdjacentPointRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > * PointId: 1 * AdjacentPointId: 2   An exception is not raised by the test case. |
| 1. Search the AdjacentPointRepository by calling AdjacentPointRepository.Find with the following Search Criteria:  * PointId: 1 * AdjacentPointId: 2 | The AdjacentPointRepository Find method returns exactly one match. The match has the following information:   * Id: < Automatically Assigned ID from the Database > * PointId: 1 * AdjacentPointId: 2   An exception is not raised by the test case. |
| 1. Remove the added entry using AdjacentPointRepository.Remove | The AdjacentPointRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->AdjacentPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestFindWithAllSearchCriteria test results | The TestFindWithAllSearchCriteria test should have a green check mark and be reported as ‘Success’. |  |

##### TestFindById Test Case

###### Description

The test verifies that adjacent track points can be found when searching by the primary key (or ID) associated with the target entry in the adjacent track point table.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a AdjacentPointRepository class instance. | Junit creates a AdjacentPointRepository instance. An exception is not raised by the test case. |
| 1. Create a AdjacentPoint class instance with the following values:  * PointId: 1 * AdjacentPointId: 2 | An adjacent track point entry instance is created with the specified values. |
| 1. Add the AdjacentPoint class instance to the AdjacentPoint Repository by calling AdjacentPointRepository.Add | The adjacent track point entry is added to the database. The AdjacentPointRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > (Referred to as ID1) * PointId: 1 * AdjacentPointId: 2   An exception is not raised by the test case. |
| 1. Search the AdjacentPointRepository by calling AdjacentPointRepository.Find with the following Parameters:   Id: <ID1> | The AdjacentPointRepository Find method returns exactly one match. The match has the following information:   * Id: <ID1> * PointId: 1 * AdjacentPointId: 2   An exception is not raised by the test case. |
| 1. Remove the added entry using AdjacentPointRepository.Remove | The AdjacentPointRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->AdjacentPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestFindById test results | The TestFindById test should have a green check mark and be reported as ‘Success’. |  |

##### TestAdd Test Case

###### Description

The test verifies that new adjacent track point entries are correctly added to the adjacent track point table.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a AdjacentPointRepository class instance. | Junit creates a AdjacentPointRepository instance. An exception is not raised by the test case. |
| 1. Create a AdjacentPoint class instance with the following values:  * PointId: 1 * AdjacentPointId: 2 | An adjacent track point entry instance is created with the specified values. |
| 1. Add the AdjacentPoint class instance to the AdjacentPoint Repository by calling AdjacentPointRepository.Add | The adjacent track point entry is added to the database. The AdjacentPointRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > * PointId: 1 * AdjacentPointId: 2   An exception is not raised by the test case. |
| 1. Compare the adjacent track point value returned by the repository from the add operation against the adjacent track point value that was added. | The two values match. The match has the following information:   * Id: < Automatically Assigned ID from the Database > * PointId: 1 * AdjacentPointId: 2   An exception is not raised by the test case. |
| 1. Remove the added entry using AdjacentPointRepository.Remove | The AdjacentPointRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->AdjacentPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestAdd test results | The TestAdd test should have a green check mark and be reported as ‘Success’. |  |

##### TestUpdate Test Case

###### Description

The test verifies that adjacent track point entries are correctly updated to the adjacent track point table.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a AdjacentPointRepository class instance. | Junit creates a AdjacentPointRepository instance. An exception is not raised by the test case. |
| 1. Create a AdjacentPoint class instance (referred to as Instance A) with the following values:  * PointId: 1 * AdjacentPointId: 2 | An adjacent track point entry instance is created with the specified values. |
| 1. Add the AdjacentPoint class instance to the AdjacentPoint Repository by calling AdjacentPointRepository.Add | The adjacent track point entry is added to the database. The AdjacentPointRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > (Referred to as ID1) * PointId: 1 * AdjacentPointId: 2   An exception is not raised by the test case. |
| 1. Change the following information for instance A using the corresponding set methods:  * AdjacentPointId: 3 | Instance A now has the following values:   * PointId: 1 * AdjacentPointId: 3 |
| 1. Update instance A’s repository entry in the AdjacentPoint repository by calling AdjacentPointRepository.Update | Instance A’s track point entry is updated in the database. The AdjacentPointRepository.Update method returns a RepositoryEntry instance with the following value:   * Id: <ID1> * PointId: 1 * AdjacentPointId: 3   An exception is not raised by the test case. |
| 1. Search for instance A by ID using AdjacentPointRepository.FindById:   ID: <ID1> | Instance A’s track point entry in the database is found with the following information:   * Id: <ID1> * PointId: 1 * AdjacentPointId: 3   An exception is not raised by the test case. |
| 1. Remove the added entry using AdjacentPointRepository.Remove | The AdjacentPointRepository removes the new entry from the database. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->AdjacentPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestUpdate test results | The TestUpdate test should have a green check mark and be reported as ‘Success’. |  |

##### TestRemove Test Case

###### Description

The test verifies that adjacent track point entries are correctly removed from the adjacent track point table. An adjacent track point entry is added, then removed, and lastly testing is done to verify that it was correctly removed by searching for the deleted item.

###### Automated Test Procedure

Procedure executed by the Junit test code.

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Connect to the Train Navigation Database | Junit connects to the Train Navigation Database. An exception is not raised by the test case. |
| 1. Create a AdjacentPointRepository class instance. | Junit creates a AdjacentPointRepository instance. An exception is not raised by the test case. |
| 1. Create a AdjacentPoint class instance (referred to as Instance A) with the following values:  * PointId: 1 * AdjacentPointId: 2 | An adjacent track point entry instance is created with the specified values. |
| 1. Add the AdjacentPoint class instance to the AdjacentPoint Repository by calling AdjacentPointRepository.Add | The adjacent track point entry is added to the database. The AdjacentPointRepository.Add method returns a RepositoryEntry instance with the following value:   * Id: < Automatically Assigned ID from the Database > (Referred to as ID1) * PointId: 1 * AdjacentPointId: 2   An exception is not raised by the test case. |
| 1. Remove the AdjacentPoint instance A from the AdjacentPointRepository using the it’s repository entry ID. | The row in the TrackBlock Database is removed. (i.e. The adjacent track point entry in the repository is removed). |
| 1. Search for the deleted repository entry using the repository entry ID: ID1. | No matches are found in the repository (i.e. database) in the search.  An exception is not raised by the test case. |
| 1. Verify that the test passed. | The Eclipse Junit Runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests ->AdjacentPointRepositoryTests.java file. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestRemove test results | The TestRemove test should have a green check mark and be reported as ‘Success’. |  |

#### TrackSwitchRepository Tests

#### AccelerometerMeasurementRepository Tests

#### GyroscopeMeasurementRepository Tests

#### RfidTagDetectedNotificationRepository Tests

## Train Navigation Service Tests

### Purpose

The purpose of this section to describe the series of tests necessary to verify that all of the requirements for the Train Navigation Service subsystem of Train Trax are met. The Train Navigation Service is a network service that contains all of the business logic necessary to transform measurements from the train (taken by the MDU) and knowledge about the track (Train Navigation Database) to estimates of the position of the train. It is also responsible for controlling switches on the track that the trains move along. The Navigation Service that is used by other Train Trax subsystems (primarily the GUI) to control and/or monitor active components on the rail system. It is not responsible for controlling the speed of the train.

### Description

Testing is conducted in accordance to the Train Trax Test Plan. Testing for this subsystem is primarily conducted through a series of integration tests created to run from the Junit unit testing framework. However, some tests are also conducted by a TestDriver console program that guides the Tester through a series of steps for testing operations that require a human to be in the loop.

Test Results from the Test Driver and the automated integration tests should be recorded as the primary artifact for verification that testing requirements work correctly. The Junit plugin from Eclipse is issued to launch the tests directly from the source projects.

The subsections that follow provide detailed information on how to conduct testing for the Train Navigation Service subsystem.

### Required Tools

* JUnit 4
* Eclipse Mars (4.5)
* My SQL Server (5.5.47)
* Java 7
* MYSQL JDBC Driver (Connector/J 5.1.38)
* JMRI 3.8 (or higher)
* Spreadsheet Program (Microsoft Excel or equivalent)
* Test Computer
* Digitrax PR3 Programming Interface attached to the Positive Test Control Test Bed
* XBee USB Adapter (Bridge used by the PC to talk to the XBees on the network)

### General Test Procedures

#### Using the Test Case/Test Procedure Tables

For the remainder of this testing document, it is expected that the Test Operator do the following:

1. Perform the actions described in the Action Column of each Test Case/Test Procedure table.
2. Determine if the behavior / output described in the ‘Expected Result’ Column happens after executing the corresponding action.
3. If the ‘Expected Result’ is observed, the fill the ‘Pass/Fail’ column with Pass. Otherwise, fill the ‘Pass/Fail’ column with Fail.

#### Open A Train Trax Workspace

Test sequence describes what is necessary to get the eclipse configured to run testing for the Train Navigation Database.

|  |  |  |
| --- | --- | --- |
| Description | Expected Result | Pass/Fail |
| 1. Open Eclipse | The eclipse application should appear upon the screen. |  |
| 1. Examine the Project Explorer window. | All of the Train Trax Projects in the CPE656TL/source directory should appear in the Project Explorer in the left hand corner of the table.  No further steps are necessary if projects are present.  If they are not present, then proceed to the next step. |  |
| 1. Click on File->Import. | The Import window should appear. |  |
| 1. Click on ‘Existing Projects into Workspace’, then click Next. | The Import window should update to the ‘Import Projects’ screen. |  |
| 1. Click on the Browse button to the right of the ‘Select root directory’ text box. | A Browse window should appear. |  |
| 1. Browse to the location of the CPE656TL/source directory. Click OK. | A list of Train Trax Projects should appear in the Projects List Box. |  |
| 1. Click on the ‘Select All’ button. | All of the Train Trax Projects should have a check mark beside their listing in the Projects List Box. |  |
| 1. Click on the Finish Button. | All of the Train Trax Projects in the CPE656TL/source directory should appear in the Project Explorer in the left hand corner of the table. |  |

#### Running Integration Tests

A description is given on how to use Eclipse to run all of the integration tests defined for the Train Navigation Database.

|  |  |  |
| --- | --- | --- |
| Description | Expected Result | Pass/Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test sequence. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationDatabase.Library.UnitTests project. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the results from the Junit tab. | All of the tests are run. This is indicated by the following ratio: (Tests Run / Tests Available). For example, if there are 83 tests then, it should report in the Runs field “83/83”.  The Errors field reports “0”.  The Failures field reports “0”. |  |

#### Running the Test Driver

A description is given on how to run the Train Navigation Database Test Driver to complete human-in the-loop testing.

|  |  |  |
| --- | --- | --- |
| Description | Expected Result | Pass/Fail |
| 1. Connect the XBee module to the test computer. | The device is recognized by the OS and is assigned a COM port. |  |
| 1. Connect the PR3 programming interface to the test computer | The device is recognized by the OS and is assigned a COM port. |  |
| 1. Go to the tools folder. Located at “<Train Trax Root>/tools” | A file browser is open showing the contents of “<Train Trax Root>/tools” |  |
| 1. Edit “navigation\_service\_test\_driver.bat” | The batch file appears in an editor. |  |
| 1. Change the “—pr3-port” argument in the call to the test driver to the COM port that the PR3 programming interface is using. | The value associated with the “—pr3-port” argument has been changed to the value assigned to the PR3 programming interface. |  |
| 1. Change the “—mdu-port” argument in the call to the test driver to the COM port that the USB XBee device. | The value associated with the “—mdu-port” argument has been changed to match the value assigned to the USB XBee device. |  |
| 1. Save “navigation\_service\_test\_driver.bat”. | The changes to ““navigation\_service\_test\_driver.bat” have been saved to file. |  |
| 1. Close the editor displaying ““navigation\_service\_test\_driver.bat” | The editor displaying “navigation\_service\_test\_driver.bat” has been closed. |  |
| 1. Return to the file browser showing “<Train Trax Root>/tools” and double click on “navigation\_service\_test\_driver.bat”. | A console window appears. |  |
| 1. Follow the steps described by the console window. | A sequence of tests is conducted where the program prompts the operator for input at stages.  When testing is complete, the program should return the following: A prompt reporting “Testing Complete” in the console window, the file path for the test results file, and the final outcome of the test driver testing: “Pass” or “Fail”. |  |

### Test Cases

#### Train Position Algorithm Tests

In order to be able to determine if the train navigation service is correctly estimating train position data, we need to have a system where the position of the train is known ahead of time so that the estimates reported by the train navigation service can be compared against some truth of where the train actually is. This is accomplished by creating a model for where the train is expected to be over a period of time.

The model takes in the initial position, orientation, initial velocity, and acceleration defined over time to calculate the final position at a given time from the kinematics formulas from physics (defined in the Design Document). Since the position estimation algorithm used by the Train Navigation Service is the only thing necessary to test the train navigation service reporting, the inputs and outputs from the class that encapsulates the algorithm (Train2DPositionAlgorithm) is what's tested. Output from the model (MduGenerator) is used to match the same format that is used by the class to read in measurements from MDU so that the tests can accurately simulate incoming measurements from a train. Outputs from the model include initial position, orientation, and velocity that was reported as input and reports of acceleration and position truth over time (at some pre-determined sample rate).

Alternatively, instead of using the kinematics formula for calculating simulated train position values. Actual Measurements from previous runs of the MDU that are captured to file may be used. This provides further confirmation that the train position algorithm should operate correctly with live MDU data. In this situation, any expected values for train position are included in the capture file and were provided by an expert that determined an approximation of the position of the train based on video captured during the collection of measurements.

##### TestCalculatePositionWithStraightLineAt45DegreeAngle Test Case

###### Description

The test uses simulated data of a train moving along the Position Train Control Test Bed at a 45 degree angle from the origin of the coordinate system to verify that the train position algorithm can correctly resolve a train position with ideal IMU and RFID position updates where there are no angular changes being made. IMU data is timestamped as if it is reported every second. RFID position updates are timestamped as if they were reported every four seconds.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Load the test case sample file located as follows from the directory that houses the unit tests: “./Test\_Cases/Simulated\_StraightLine\_   From\_0\_0\_45Degree\_1mps\_0mpsq\_pt35mpsq.csv” | All of the information needed for the desired test case is loaded. The initial position is at the origin. The orientation is 45 degrees from pointing on the x-axis in the first quadrant where the rotation is along the z-axis of the coordinate system (along the height of the table, perpendicular to the floor). The train is at rest for 50 samples to complete calibration. A kinetic friction offset of 0.35 (m/s^2) has been injected into the accelerometer values.  After 50 samples, then the train accelerates (a 1 m/s^2) for 1 second, then moves at the expected speed result speed (1 m/s) in a straight line. |
| 1. Initialize the Train Position Algorithm. | Junit creates a Train2DPositionAlgorithm instance where the initial orientation is 45 degrees along the z-axis The initial position is at the origin. An exception is not raised by the test case. |
| 1. Call the train position algorithm to calculate the train’s position from the measurements for a single collection of samples, taken from the loaded test case, that represent measurements taken at an instant in time. This is intended to simulate a single update reported by the MDU. | A new estimate on the train’s position is calculated. |
| 1. Verify that the train position estimate is correct. Compare the train position reported by the algorithm against the expected position calculated for the data at that point in time (calculated when the simulated data was generated) | The position reported by the algorithm matches the expected position calculated for the point in time.  An exception is not raised by the test case. |
| 1. If this there are any measurements left, then go back to step 3. Otherwise, the test is completed. | If there are any measurements left, then test execution continues at step 3. Otherwise the Eclipse Junit runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationService.Library.UnitTests project. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestCalculatePositionWithStraightLineAt45DegreeAngle test results | The ‘TestCalculatePosition  WithStraightLineAt45DegreeAngle’ test should have a green check mark and be reported as ‘Success’. |  |

##### TestCalculatePositionWithCircleAtTableCenter Test Case

###### Description

The test uses simulated data of a train moving along the Position Train Control Test Bed in a circle from the center of the table with a radius of 2 ft (equals circumference of 7.660 meters) to verify that the train position algorithm can correctly resolve a train position with ideal IMU and RFID position updates where there are a lot of angular changes being made..IMU data is timestamped as if it is reported every second. RFID position updates are timestamped as if they were reported every four seconds. The train is constantly moving at 1 m/s. The should equal an angular rate change of about 46.997 degrees per second => (0.82 radians per second) as it moves along the circle.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Load the test case sample file located as follows from the directory that houses the unit tests: “./Test\_Cases/Simulated\_Circle\_2ftRadius\_   From\_108\_48\_0Degree\_1mps\_0mpsq\_pt35mpsq.csv” | All of the information needed for the desired test case is loaded. The initial position is at the center of the table (108 inches, 48 inches). The orientation is 0 degrees from pointing on the x-axis in the first quadrant where the rotation is along the z-axis of the coordinate system (along the height of the table, perpendicular to the floor). The train is at rest for 50 samples to complete calibration. A kinetic friction offset of 0.35 (m/s^2) has been injected into the accelerometer values.  After 50 samples, then the train accelerates (a 1 m/s^2) for 1 second, then moves at the expected speed result speed (1 m/s) in a 2ft radius circle. |
| 1. Initialize the Train Position Algorithm. | Junit creates a Train2DPositionAlgorithm instance where the initial orientation is 0 degrees along the z-axis The initial position is at the center of the table (108 inches, 48 inches). An exception is not raised by the test case. |
| 1. Call the train position algorithm to calculate the train’s position from the measurements for a single collection of samples, taken from the loaded test case, that represent measurements taken at an instant in time. This is intended to simulate a single update reported by the MDU. | A new estimate on the train’s position is calculated. |
| 1. Verify that the train position estimate is correct. Compare the train position reported by the algorithm against the expected position calculated for the data at that point in time (calculated when the simulated data was generated) | The position reported by the algorithm matches the expected position calculated for the point in time.  An exception is not raised by the test case. |
| 1. If this there are any measurements left, then go back to step 3. Otherwise, the test is completed. | If there are any measurements left, then test execution continues at step 3. Otherwise the Eclipse Junit runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationService.Library.UnitTests project. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestCalculatePositionWithCircleAtTableCenter test results | The ‘TestCalculatePosition  WithCircleAtTableCenter’ test should have a green check mark and be reported as ‘Success’. |  |

##### TestCalculatePositionWithActualSampleData Test Case

###### Description

The test uses a sampling of raw data from a train moving along the target a figure 8 section of track in the target Positive Train Control Test Bed to verify that the train position algorithm can correctly resolve a train position with actual IMU and RFID measurements. Data was sampled from the train moving along a figure 8 track at 6 inches per second. Approximately 2 seconds worth of data is selected. During the duration of time, 3 RFID tags are crossed. The expected truth for the actual measurements was gained from visual inspection of the video sample that was taken in parallel to the sampling of the MDU measurements by correlating the video timestamps with the RFID Tag truth and the visual markers on the video: ‘Train Demo.wmv’ collected on 2/17/16. 50 samples from the beginning of the sampling of video is appended to the start of the sample in order for the measurements to be properly calibrated. Then evaluation of the algorithm happens after the 2nd RFID tag is crossed since this is where the algorithm should properly initialized. The remaining second where a traversal happens from the 2nd RFID tag to the 3rd RFID tag is what’s measured.

The actual data sample duration is only 2 seconds because error from each sampling (noise from bumps, error from the device, and error from alignment causing a proportion of gravity to be  
measured) quickly accumulate when calculating position over a long period of time with IMU data alone. Two seconds was chosen to be a long enough sample for the train to travel to the next RFID Tag under typical conditions and short enough that the worst-case average error of acceleration (0.01 m/s^2) will not overcome the average displacement (i.e. It needed to be short enough that it is realistic for us to expect the result calculated for position to be good). It’s the minimum amount of data necessary to demonstrate IMU reset after an RFID Tag is hit and that positions are calculated correctly between tag detection events.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Load the test case sample file located as follows from the directory that houses the unit tests: “./Test\_Cases/Actual\_TestTrack\_02-17-16\_From\_XInches\_YInches\_RDegree\_pt1524mps\_0mpsq\_pt35mpsq.csv” | All of the information needed for the desired test case is loaded. The initial position is at the the start of the figure 8 loop (X inches, Y inches). The orientation is R degrees from pointing on the x-axis in the first quadrant where the rotation is along the z-axis of the coordinate system (along the height of the table, perpendicular to the floor). The train is at rest for 50 samples to complete calibration. A kinetic friction offset of 0.35 (m/s^2) is assumed to be present in the accelerometer values.  After 50 samples, the actual measurements from the train at the point of interest begins and fills the remainder of the test case samples. |
| 1. Initialize the Train Position Algorithm. | Junit creates a Train2DPositionAlgorithm instance where the initial orientation is R degrees along the z-axis The initial position is at the origin. An exception is not raised by the test case. |
| 1. Call the train position algorithm to calculate the train’s position from the measurements for a single collection of samples, taken from the loaded test case, that represent measurements taken at an instant in time. This is intended to simulate a single update reported by the MDU. | A new estimate on the train’s position is calculated. |
| 1. Verify that the train position estimate is correct. Compare the train position reported by the algorithm against the expected position calculated for the data at that point in time (calculated when the simulated data was generated) | The position reported by the algorithm matches the expected position calculated for the point in time.  An exception is not raised by the test case. |
| 1. If this there are any measurements left, then go back to step 3. Otherwise, the test is completed. | If there are any measurements left, then test execution continues at step 3. Otherwise the Eclipse Junit runner reports ‘Success’ for the test. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Open A Train Trax Workspace’ test procedure. | All of the Train Trax Projects are loaded into the Eclipse workspace. |  |
| 1. Right click upon the TrainNavigationService.Library.UnitTests project. | A context menu appears. |  |
| 1. Click on Run As -> Junit Test | The Junit tab should appear and all of the tests for the database should be run. |  |
| 1. Examine the test results in the Junit tab on the Eclipse Window for the TestCalculatePositionWithActualSampleData test results | The ‘TestCalculatePosition  WithActualSampleData’ test should have a green check mark and be reported as ‘Success’. |  |

##### Verify Train Movement Test Case

###### Description

The test uses a sampling of raw data from a train moving along the target a figure 8 section of track in the target Positive Train Control Test Bed to verify that the train position algorithm can correctly resolve a train position for the entire traversal of a loop of track. Data was sampled from the train moving along a figure 8 track at 6 inches per second. The speed of 6 inches per second is consistent with the train used for the measurement operating at full throttle. The speed of the train was determined by observing from captured video associated with the sample, the average period of time between each detection of an RFID tag along a test track of the Positive Train Control Test Bed. For this track, RFID tags were placed 6 inches apart from each other and it took approximately 1 second between tag detections. A LED was mounted on top of the train to indicate when each RFID tag was crossed.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Load the test case sample file located as follows from the directory that houses the unit tests: “./Test\_Cases/Actual\_TestTrack\_02-17-16.csv” | All of the information needed for the desired test case is loaded. All of the raw samples collected from the MDU when it traversed the test track on 2/17/16 are loaded as well as the initial position and orientation of the train at the start of the sampling. |
| 1. Initialize the Train Position Algorithm. | Junit creates a Train2DPositionAlgorithm instance where the initial position and orientation are assigned to the values specified in the loaded test case data. |
| 1. Call the train position algorithm to calculate the train’s position from the measurements for a single collection of samples, taken from the loaded test case, that represent measurements taken at an instant in time. This is intended to simulate a single update reported by the MDU. | A new estimate on the train’s position is calculated. |
| 1. Record the train position estimates into an output CSV File with the following information: TrainId, X Position (inches), Y Position(inches), Z Position (inches), Timestamp (MM-DD-YYYY HH:mm:ss). | A file called “TrainPositionEstimates.csv” is created in a subdirectory in the working directory called “TestOutput” (i.e. filepath = “./TestOutput/  TrainPositionEstimates.csv”)  An exception is not raised by the test case. |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Running the Test Driver’ test procedure. | A console application launches with a menu prompting for a menu item selection. |  |
| 1. Select “Verify Train Movement” | A prompt appears asking for the train position test case file. |  |
| 1. Type the following and press ENTER:   “./Test\_Cases/Actual\_TestTrack\_02-17-16.csv” | A message should appear stating that train position estimates are complete. And that the position estimate file is located at “./TestOutput/  TrainPositionEstimates.csv”  It should also prompt for ENTER to be pressed. |  |
| 1. Press ENTER. | The program returns to the main menu. |  |
| 1. Select “Exit” | The console application terminates. |  |
| 1. Open “./TestOutput/TrainPositionEstimates.csv” with the Spreadsheet program | The contents of the CSV file are loaded into the spreadsheet program |  |
| 1. Create a scatter plot of the “X Position (inches)” and “Y Position (inches)” columns with points and lines. | A scatter plot is created where there is the shape of a figure 8. For the figure 8, (min Y: 15 inches, max Y: 88 inches, min X: 10 inches, max X: 47 inches) This is consistent with the train moving along the geometry of the track in this sample. |  |

##### Verify Track Switch Control Test Case

###### Description

The test connects to the PR3 programming interface and uses it as a LocoNet message bridge to control switches on the test track.

###### Automated Test Procedure

|  |  |
| --- | --- |
| **Automated Test Step** | **Expected Result** |
| 1. Initialize the Train Navigation Service. | The TestDriver creates a TrainNavigationService instance where the initial position is at the origin and the orientation along the z-axis is 0 Degrees. The Service is also configured to communicate with the PR3 programming interface according to the settings specified in the program arguments of the test driver. |
| 1. A call is made to the Train Navigation Service to change the requested switch into the ByPass state. | A LocoNet request switch message is sent to the DS64 switches on the Positive Train Control Test bed to place the switch into the Thrown State. The addressing should match the decimal value of the switchNumber minus “SW” minus 1. (e.g. “SW43” equals a decimal value address of 42) |
| 1. Test Waits for Operator to Press a Key to Continue | Test does not progress until the Test Operator Presses a Key. |
| 1. A call is made to the Train Navigation Service to change the requested switch into the Pass state. | A LocoNet request switch message is sent to the DS64 switches on the Positive Train Control Test bed to place the switch into the Closed State. The addressing should match the decimal value of the switchNumber minus “SW” minus 1. (e.g. “SW43” equals a decimal value address of 42) |

###### Test Operator Test Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in the ‘Running the Test Driver’ test procedure. | A console application launches with a menu prompting for a menu item selection. |  |
| 1. Select “Verify Track Switch Control” | A prompt appears requesting a switch number. |  |
| 1. Type the following and press ENTER:   SW43 | The switch labeled ‘43’ on the Positive Test Control Test Bed is in the thrown state. The TestDriver displays a prompt stating that the requested switch is now in the ByPass state and requests for ENTER to be pressed to continue. |  |
| 1. Press ENTER. | The switch labeled ‘43’ on the Positive Test Control Test Bed is changed to the closed state. The TestDriver displays a prompt stating that the requested switch is now in the Pass state and prompts for ENTER to be pressed. |  |
| 1. Press ENTER. | The program returns to the main menu. |  |
| 1. Select “Exit” | The console application terminates. |  |

## Train Navigation GUI Tests

### Purpose

The purpose of this section to describe the series of tests necessary to verify that all of the requirements for the Train Navigation GUI subsystem of Train Trax are met. The Train Navigation GUI is the primary interface that Train Operators use to control the system. It responsible for rendering the geometry of the Positive Train Control Bed from the data provided by the Train Navigation Database and to continuously report on the state of active components of the rail system (in particular the trains and switches).

### Description

Testing is conducted in accordance to the Train Trax Test Plan. Testing for this subsystem is primarily through the test operator manipulating the UI (either from an Android emulator or device) and comparing the information that it reports against the actual state of the test bed.

The subsections that follow provide detailed information on how to conduct testing for the Train Navigation GUI subsystem.

### Required Tools

* Android Studio
* Android Emulator (or an Android Mobile Device)
* Test Computer Running the Train Navigation Service Rest Service (which is connected to a PR3 Programming Interface that is connected to the Positive Train Control Test Bed)
* Test Computer Running the Train Navigation Database Rest Service (With information about the Positive Train Control Test Bed measurements)

### Android Train Trax GUI Test Procedures

#### Using the Test Case/Test Procedure Tables

For the remainder of this testing document, it is expected that the Test Operator do the following:

1. Perform the actions described in the Action Column of each Test Case/Test Procedure table.
2. Determine if the behavior / output described in the ‘Expected Result’ Column happens after executing the corresponding action.
3. If the ‘Expected Result’ is observed, the fill the ‘Pass/Fail’ column with Pass. Otherwise, fill the ‘Pass/Fail’ column with Fail.

#### Run Train Trax from Android Studio

Test sequence describes what is necessary to run Train Trax From Android Studio

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass/Fail |
| 1. Open Android Studio | The Android Studio application should appear upon the screen. |  |
| 1. Go to File Menu and Select Open | A Project Explorer should appear in the middle of the screen. Go to the folder where the Train Trax project is stored on the computer and then select Open to open the project. A green android studio symbol will appear next to valid project files that you can select. |  |
| 1. Build Project by going under Build Menu and Selecting Make Project. | Under the messages tab at the bottom you should see a BUILD SUCCESSFUL message with 0 errors. |  |
| 1. Connect a tablet device to the computer with a usb or appropriate connection type | A message should pop up saying that the computer recognizes the device, USB debugging must be enabled if you wish to run in debug mode. |  |
| 1. Under the Run menu select Run App | Verify that your device shows up in the Device Chooser Display as a running device |  |
| 1. Select Device and Select ok to run the app on your device. | The App should start up and run. |  |
| 1. Go to the settings menu and select Edit Port | Enter the port number to connect to the Train Navigation Service |  |
| 1. Go to the settings menu and select Edit IP address | Enter the IP address to connect to the Train Navigation Service |  |

#### Installing and Running Train Trax without Android Studio

Test sequence describes what is necessary to run Train Trax directly from an Android device

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass/Fail |
| 1. Download the app .apk file onto device to install it. | .apk file should be stored in the Downloads folder |  |
| 1. Select the APK file from the Downloads folder. | An option to open with Package Installer will be presented. |  |
| 1. Select Open with Package Installer | A window with an option to install Train Trax will be presented. |  |
| 1. Select Install | Installation Process is initiated and App Is installed |  |
| 1. Open app from the Apps Menu | Train Trax App should startup and run. |  |

### Test Cases

TestGuiMainMenuDisplay

TestGuiMainMenuErrorDisplay

TestGuiTrainMonitorDisplay

TestSwitchDisplay

TestTrainPostitionDisplay

#### TestGuiMainMenuDisplay Test Case

Description

The test verifies that Main Menu for the Android Train Trax App loads and displays all of the appropriate information. This case assumes that the Train Navigation Database is up and running and has the required information to display the track diagram.

Test Operator Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in Running App from Android Studio or Installing and Running App on device | The Android Application starts up without error. |  |
| 1. Check to see if the title bar is there | Should display “Train Trax System” |  |
| 1. Check the diagram of the track drawn | Should be consistent with the shape of the track. |  |
| 1. Check to make sure the Monitor Button is at the top right corner of the screen and not ghosted | Should be a blue button that says “Monitor Train” |  |

#### TestGuiMainMenuErrorDisplay Test Case

Description

The test verifies that Main Menu for the Android Train Trax App loads and displays all of the appropriate information. This case assumes that the Train Navigation Database is not up and running or does not have the required information to display the track diagram.

Test Operator Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in Running App from Android Studio or Installing and Running App on device | The Android Application starts up without error. |  |
| 1. Check to see if the title bar is there | Should display “Train Trax System” |  |
| 1. Check to make sure there an error message indicating that we are unable to load the track data. | Screen should be blank with an error message. |  |
| 1. Check to make sure the Monitor Button is at the top right corner of the screen is ghosted (clicking it does nothing). | Should be a blue button that says “Monitor Train” |  |

#### TestGuiTrainMonitorDisplay Test Case

Description

The test verifies that the Train Monitor View for the Android Train Trax App loads and displays all of the appropriate information.

Test Operator Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in Running App from Android Studio or Installing and Running App on device | The Android Application starts up without error. |  |
| 1. From Main Menu Select Monitor Train Button | Should display a new view screen with the title Monitor Train. |  |
| 1. Check the diagram of the track drawn | Should be consistent with the shape of the track. |  |
| 1. Check color of track diagram | The active path controlled by the switch should be green and in the incorrect path should be colored red. |  |
| 1. Check to make sure the track switches are displayed | Should be square boxes that say switch colored red or black depending on the state. |  |
| 1. Check to make sure a legend appears at the bottom right corner of the screen. | Legend displays the meaning of the colors and symbols displayed on the diagram. |  |

#### TestGuiSwitchDisplay Test Case

Description

The test verifies that Track switches In the Train Monitor View for the Android Train Trax App function correctly, and displays all of the correct information.

Test Operator Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in Running App from Android Studio or Installing and Running App on device | The Android Application starts up without error. |  |
| 1. Click on the Monitor Train Button | Train Monitor View should be displayed. |  |
| 1. Check the state of each switch to see if it matches the switch states on the actual track. | The train navigation service should sent the initial state of the switches upon startup. |  |
| 1. Click on a switch | Switch color should change either from black to red or red to black depending on the state it’s currently in. |  |
| 1. Verify that the display of the track diagram state has been updated to reflect the switch state change. | The segments of the track diagram that are connected to the switch should switch colors/state. The active segment should be green and the inactive should be red. |  |
| 1. Verify that when a switch is clicked the information is sent and received by the Train Navigation Service. | The switch state of the train diagram should be the opposite of the previous state and should be reflect on the actual track diagram. |  |

#### TestGuiTrainPositionDisplay Test Case

Description

The test verifies that Main Menu for the Android Train Trax App loads and displays all of the appropriate information.

Test Operator Procedure

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Pass /Fail |
| 1. Perform the steps described in Running App from Android Studio or Installing and Running App on device | The Android Application starts up without error. |  |
| 1. Click on the Monitor Train Button | Should display “Train Trax System” | Train Monitor View should be displayed. |
| 1. Verify an icon is displayed representing a train. | Should be a blue icon with the word “train”. |  |
| 1. Verify the number of train icons displayed is correct and the position displayed of those train icons is correct. | This information should be received by the Train Navigation Service. |  |
| 1. Verify the train position updates every second. | Icon should move to show the latest calculated position of the train. |  |

# Requirements Traceability

## Traceability from Test Case to requirement

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Test Case Name | Requirement Number | Requirement text |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Train Navigation Database

#### Integration Tests

These are a series of tests that test the complete implementation of the Train Navigation Database as it is used by clients of the library. The intent of these tests are to verify that all public interfaces to the subsystem function correctly prior to any effort of clients to integrate with the library (hence the term integration tests).

##### Track Geometry

The Train Navigation Database is responsible for recording information about the size and shape of the rail road tracks that consist of the Positive Train Control Test Bed. This is accomplished by storing information about the train track using a bottom-up hierarchy of tables where each table represents a component that is a fundamental building block of the track. The smallest building block are track points. These are individual measurements of important positions on the test bed track. Positions are ensured to be recorded at a fixed distance to ensure a certain level of fidelity for representing the shape of the points alone. Positions are measured in inches relative to a point on the track that acts as the origin for the coordinate system. The origin is the bottom, left-most corner at the top of table that the Test Bed sits upon. The depth from the origin represents the y coordinates. The width from the origin represents the x coordinates, and the height from the origin represents the z coordinates. All of the points are stored in the Track Points tables.

Information about the proximity of a given point to other points is captured with the Adjacent Points table. It records which points are neighbors to other points. Recording adjacent points allows each point to be treated as vertex in a graph and the Adjacent Points table acts as the adjacency list for that graph.

Prior to Train Trax, the system was organized into AdjacentPoints where each block of the track is a section of the track that has been wired so that current from the DCC signals could be used to detect whether or not trains are present in a given section (DCC AdjacentPoint Occupancy Detection). The AdjacentPoint table is used to record which Track Points correspond to one of these track blocks. A collection of AdjacentPoints is what is used to represent the entire track.

There are switches used on the track to control the movement on trains between different routes on the rail system. In practice, a track switch controls the next track block that a train enters once it crosses the switch. The Train Switch table is used to record the position of each switch on the track as well as the AdjacentPoints that it is responsible for controlling access to.

The tests that follow are used to verify that the Train Navigation Database is correctly controlling each Track Geometry table described.

###### AdjacentPoint Repository

|  |  |  |
| --- | --- | --- |
| **Junit Integration Test Method** | **Description** | **Met Requirements** |
| TestFindWithAllSearchCriteria | The test verifies that track blocks can be found when searches against all of the values of the columns in the track block table are performed. | TND-3020 |
| TestFindWithBlockNameSearchCriteria | The test verifies that track blocks can be found according to the human friendly name associated with it. | TND-3020 |
| TestFindWithEmptySearchCriteria | The test verifies that a search defaults to a listing of all track block values if there are not any values specified to find matches for. | TND-3020 |
| TestFindAll | The test verifies that a listing of all stored values can be provided. | TND-3020 |
| TestRemove | The test verifies that track blocks can be removed if erroneous block entries are identified. | TND-3020 |
| TestFindById | The test verifies that track blocks can be found by the unique ID associated with each entry into the repository. | TND-3020 |
| TestUpdate | The test verifies that existing track block entries can be changed. | TND-3020 |
| TestUpdateWithInvalidId | The test verifies that the repository handles when an update to a non-existent entry is requested. | TND-3020 |
| TestRemoveWithInvalidId | The test verifies that the repository handles when removal of a non-existent entry is requested. | TND-3020 |
| TestFindWithInvalidId | The test verifies that the repository handles when searching for a non-existent entry is requested. | TND-3020 |
| TestAdd | The test verifies that track blocks can be added to the repository. | TND-3020 |

###### Track Point Repository

|  |  |  |
| --- | --- | --- |
| **Junit Integration Test Method** | **Description** | **Met Requirements** |
| TestFindWithAllSearchCriteria | The test verifies that track points can be found when searches against all of the values of the columns in the track point table are performed. This functionality is necessary for a client to be able to calculate orientation for a track point: We need absolute position of points. | TND-3010, TND-4011, TND-3060 |
| TestFindWithNameSearchCriteria | The test verifies that track points can be found according to the human friendly name associated with it. | TND-3010, TND-4011 |
| TestFindWithBlockIdSearchCriteria | The test verifies that track points can be found according to the repository ID for the track block associated with the point. | TND-3010, TND-4011 |
| TestFindWithTagNameSearchCriteria | The test verifies that track points can be found according to the RFID Tag identifier associated with it. | TND-3010, TND-4011 |
| TestFindWithTypeSearchCriteria | The test verifies that track points can be found according to the point type associated with it. A point type is a text description of a group of points used for similar purpose. | TND-3010, TND-4011 |
| TestFindWithEmptySearchCriteria | The test verifies that a search defaults to a listing of all track point values if there are not any values specified to find matches for. | TND-3010, TND-4011 |
| TestFindAll | The test verifies that a listing of all stored values can be provided. | TND-3010, TND-4011 |
| TestRemove | The test verifies that track points can be removed if erroneous point entries are identified. | TND-3010, TND-4011 |
| TestFindById | The test verifies that track points can be found by the unique ID associated with each entry into the repository. This functionality is necessary for a client to be able to calculate orientation for a track point: We need absolute position of points. | TND-3010, TND-4011, TND-3060 |
| TestUpdate | The test verifies that existing track point entries can be changed. | TND-3010, TND-4011 |
| TestUpdateWithInvalidId | The test verifies that the repository handles when an update to a non-existent entry is requested. | TND-3010, TND-4011 |
| TestRemoveWithInvalidId | The test verifies that the repository handles when removal of a non-existent entry is requested. | TND-3010, TND-4011 |
| TestFindWithInvalidId | The test verifies that the repository handles when searching for a non-existent entry is requested. | TND-3010, TND-4011 |
| TestAdd | The test verifies that track points can be added to the repository. | TND-3010, TND-4011 |

###### Adjacent Point Repository

|  |  |  |
| --- | --- | --- |
| **Junit Integration Test Method** | **Description** | **Met Requirements** |
| TestFindWithAllSearchCriteria | The test verifies that adjacent track points on the track can be found when searches against all of the values of the columns in the adjacent point table are performed. This functionality is necessary for a client to be able to calculate orientation for a track point: We need the next point that the train will cross in order to determine the change in heading. | TND-4000, TND-3060 |
| TestFindWithAdjacentPointIdSearchCriteria | The test verifies that adjacent track points to a track point of interest can be found according to the Track Point Repository ID associated with the neighbors. | TND-4000, TND-3060 |
| TestFindWithPointIdSearchCriteria | The test verifies that adjacent track points to a track point of interest can be found according to the Track Point Repository ID associated with the track point of interest.  We need the next point that the train will cross in order to determine the change in heading. | TND-4000, TND-3060 |
| TestFindWithEmptySearchCriteria | The test verifies that a search defaults to a listing of all adjacent track point entry values if there are not any values specified to find matches for. | TND-4000, TND-3060 |
| TestFindAll | The test verifies that a listing of all stored values can be provided. | TND-4000, TND-3060 |
| TestRemove | The test verifies that an adjacency relationship between two track points can be removed if erroneous point entries are identified. | TND-4000, TND-3060 |
| TestFindById | The test verifies that adjacency association between two track points can be found by the unique ID associated with each entry into the repository. | TND-4000, TND-3060 |
| TestUpdate | The test verifies that existing adjacent track point entries can be changed. | TND-4000, TND-3060 |
| TestUpdateWithInvalidId | The test verifies that the repository handles when an update to a non-existent entry is requested. | TND-4000, TND-3060 |
| TestRemoveWithInvalidId | The test verifies that the repository handles when removal of a non-existent entry is requested. | TND-4000, TND-3060 |
| TestFindWithInvalidId | The test verifies that the repository handles when searching for a non-existent entry is requested. | TND-4000, TND-3060 |
| TestAdd | The test verifies that an adjacency relationship between track points can be added to the repository. | TND-4000, TND-3060 |

###### Track Switch Repository

|  |  |  |
| --- | --- | --- |
| **Junit Integration Test Method** | **Description** | **Met Requirements** |
| TestFindWithAllSearchCriteria | The test verifies that track switches can be found when searches against all of the values of the columns in the track switch table are performed. | TND-3030 |
| TestFindWithSwitchNameSearchCriteria | The test verifies that track switches can be found according to the human friendly name associated with it. | TND-3030 |
| TestFindWithPassBlockIdSearchCriteria | The test verifies that track switches can be found according to the repository ID for the track block that the switch directs traffic to when in Pass mode. | TND-3030 |
| TestFindWithBypassBlockIdSearchCriteria | The test verifies that track switches can be found according to the repository ID for the track block that the switch directs traffic to when in ByPass mode. | TND-3030 |
| TestFindWithPointIdSearchCriteria | The test verifies that track switches can be found according to the ID in the track point repository that stores the position of the switch. | TND-3030 |
| TestFindWithEmptySearchCriteria | The test verifies that a search defaults to a listing of all track switch values if there are not any values specified to find matches for. | TND-3030 |
| TestFindAll | The test verifies that a listing of all stored values can be provided. | TND-3030 |
| TestRemove | The test verifies that track switches can be removed if erroneous switch entries are identified. | TND-3030 |
| TestFindById | The test verifies that track switches can be found by the unique ID associated with each entry into the repository. | TND-3030 |
| TestUpdate | The test verifies that existing track switch entries can be changed. | TND-3030 |
| TestUpdateWithInvalidId | The test verifies that the repository handles when an update to a non-existent entry is requested. | TND-3030 |
| TestRemoveWithInvalidId | The test verifies that the repository handles when removal of a non-existent entry is requested. | TND-3030 |
| TestFindWithInvalidId | The test verifies that the repository handles when searching for a non-existent entry is requested. | TND-3030 |
| TestAdd | The test verifies that track switches can be added to the repository. | TND-3030 |

##### Train Measurements

[TBD]

###### Accelerometer Measurement Repository (Incomplete)

|  |  |  |
| --- | --- | --- |
| **Junit Integration Test Method** | **Description** | **Met Requirements** |
| TestFindWithAllSearchCriteria | The test verifies that accelerometer measurements can be found when searches against all of the values of the columns in the accelerometer measurements table are performed. | TND-2010 |
| TestFindWithEmptySearchCriteria | The test verifies that a search defaults to a listing of all accelerometer measurement values if there are not any values specified to find matches for. | TND-2010 |
| TestFindAll | The test verifies that a listing of all stored values can be provided. | TND-2010 |
| TestRemove | The test verifies that accelerometer measurements can be removed if erroneous measurement entries are identified. | TND-2010 |
| TestFindById | The test verifies that accelerometer measurements can be found by the unique ID associated with each entry into the repository. | TND-2010 |
| TestUpdate | The test verifies that existing accelerometer measurement entries can be changed. | TND-2010 |
| TestUpdateWithInvalidId | The test verifies that the repository handles when an update to a non-existent entry is requested. | TND-2010 |
| TestRemoveWithInvalidId | The test verifies that the repository handles when removal of a non-existent entry is requested. | TND-2010 |
| TestFindWithInvalidId | The test verifies that the repository handles when searching for a non-existent entry is requested. | TND-2010 |
| TestAdd | The test verifies that accelerometer measurements can be added to the repository. | TND-2010 |

###### Gyroscope Measurement Repository (Incomplete)

|  |  |  |
| --- | --- | --- |
| **Junit Integration Test Method** | **Description** | **Met Requirements** |
| TestFindWithAllSearchCriteria | The test verifies that gyroscope measurements can be found when searches against all of the values of the columns in the gyroscope measurements table are performed. | TND-2020 |
| TestFindWithEmptySearchCriteria | The test verifies that a search defaults to a listing of all gyroscope measurement values if there are not any values specified to find matches for. | TND-2020 |
| TestFindAll | The test verifies that a listing of all stored values can be provided. | TND-2020 |
| TestRemove | The test verifies that gyroscope measurements can be removed if erroneous measurement entries are identified. | TND-2020 |
| TestFindById | The test verifies that gyroscope measurements can be found by the unique ID associated with each entry into the repository. | TND-2020 |
| TestUpdate | The test verifies that existing gyroscope measurement entries can be changed. | TND-2020 |
| TestUpdateWithInvalidId | The test verifies that the repository handles when an update to a non-existent entry is requested. | TND-2020 |
| TestRemoveWithInvalidId | The test verifies that the repository handles when removal of a non-existent entry is requested. | TND-2020 |
| TestFindWithInvalidId | The test verifies that the repository handles when searching for a non-existent entry is requested. | TND-2020 |
| TestAdd | The test verifies that accelerometer measurements can be added to the repository. | TND-2020 |

###### RFID Detected Notification Repository (Incomplete)

|  |  |  |
| --- | --- | --- |
| **Junit Integration Test Method** | **Description** | **Met Requirements** |
| TestFindWithAllSearchCriteria | The test verifies that RFID tag detected notifications can be found when searches against all of the values of the columns in the track switch table are performed. | TND-2030, TND-2031, TND-2032 |
| TestFindWithEmptySearchCriteria | The test verifies that a search defaults to a listing of all RFID tag detected notification values if there are not any values specified to find matches for. | TND-2030, TND-2031, TND-2032 |
| TestFindAll | The test verifies that a listing of all stored values can be provided. | TND-2030, TND-2031, TND-2032 |
| TestRemove | The test verifies that RFID tag detected notifications can be removed if erroneous notification entries are identified. | TND-2030, TND-2031, TND-2032 |
| TestFindById | The test verifies that RFID tag detected notifications can be found by the unique ID associated with each entry into the repository. | TND-2030, TND-2031, TND-2032 |
| TestUpdate | The test verifies that existing RFID tag detected notification entries can be changed. | TND-2030, TND-2031, TND-2032 |
| TestUpdateWithInvalidId | The test verifies that the repository handles when an update to a non-existent entry is requested. | TND-2030, TND-2031, TND-2032 |
| TestRemoveWithInvalidId | The test verifies that the repository handles when removal of a non-existent entry is requested. | TND-2030, TND-2031, TND-2032 |
| TestFindWithInvalidId | The test verifies that the repository handles when searching for a non-existent entry is requested. | TND-2030, TND-2031, TND-2032 |
| TestAdd | The test verifies that RFID tag detected notifications can be added to the repository. | TND-2030, TND-2031, TND-2032 |

###### Train Position Repository (Incomplete)

|  |  |  |
| --- | --- | --- |
| **Junit Integration Test Method** | **Description** | **Met Requirements** |
| TestFindWithAllSearchCriteria | The test verifies that train position estimates can be found when searches against all of the values of the columns in the track switch table are performed. | TND-1000, TND-1010, TND-1020 |
| TestFindWithEmptySearchCriteria | The test verifies that a search defaults to a listing of all train position estimate values if there are not any values specified to find matches for. | TND-1000, TND-1010, TND-1020 |
| TestFindAll | The test verifies that a listing of all stored values can be provided. | TND-1000, TND-1010, TND-1020 |
| TestRemove | The test verifies that train position estimates can be removed if erroneous estimate entries are identified. | TND-1000, TND-1010, TND-1020 |
| TestFindById | The test verifies that train position estimates can be found by the unique ID associated with each entry into the repository. | TND-1000, TND-1010, TND-1020 |
| TestUpdate | The test verifies that existing train position estimate entries can be changed. | TND-1000, TND-1010, TND-1020 |
| TestUpdateWithInvalidId | The test verifies that the repository handles when an update to a non-existent entry is requested. | TND-1000, TND-1010, TND-1020 |
| TestRemoveWithInvalidId | The test verifies that the repository handles when removal of a non-existent entry is requested. | TND-1000, TND-1010, TND-1020 |
| TestFindWithInvalidId | The test verifies that the repository handles when searching for a non-existent entry is requested. | TND-1000, TND-1010, TND-1020 |
| TestAdd | The test verifies that train position estimates can be added to the repository. | TND-1000, TND-1010, TND-1020 |

##### Test Driver

### [TBD] Table of Test Driver tests here

## Traceability Matrices

### Train Navigation Database

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Requirement | Testing Area | | | | | | | |
|  | AdjacentPoint Repository | Track Point Repository | Adjacent Point Repository | Track Switch Repository | Accelerometer Measurement Repository | Gyroscope Measurement Repository | RFID Tag Detected Notification Repository | Train Position Repository |
| TND-1000 |  |  |  |  |  |  |  | X |
| TND-1010 |  |  |  |  |  |  |  | X |
| TND-1020 |  |  |  |  |  |  |  | X |
| TND-2000 |  |  |  |  | x | x | x |  |
| TND-2010 |  |  |  |  | x |  |  |  |
| TND-2020 |  |  |  |  |  | x |  |  |
| TND-2030 |  |  |  |  |  |  | x |  |
| TND-2031 |  |  |  |  |  |  | x |  |
| TND-2032 |  |  |  |  |  |  | x |  |
| TND-3000 | x | x | x | x |  |  |  |  |
| TND-3010 |  | x |  |  |  |  |  |  |
| TND-3020 | x |  |  |  |  |  |  |  |
| TND-3030 |  |  |  | x |  |  |  |  |
| TND-3060 |  | x | x |  |  |  |  |  |
| TND-4000 |  |  | x |  |  |  |  |  |
| TND-4011 |  | x |  |  |  |  |  |  |
| TND-5000 |  |  |  |  |  |  |  |  |

# Notes

# Appendixes