Railroad Dispatching Simulator

Joe Damico

CMPS 450: Senior Project

September 23, 2019

Part I: Introduction

For this project I decided to combine two of my interests; the first obviously being computer, and more specifically web applications. And the second being railroads which I've had a love for since I was a small child. Moreover, I always found dispatching (or controlling train movements) particularly interesting, because on the surface it seems like it would be an easy job. It actually takes a lot of mental power and planning ahead to make sure every train keeps moving. The mental challenge is what made me decide to create a simulator of a real dispatching desk. I modeled the simulator off of New Jersey Transits Main Line Desk, which controls the tracks between Laurel (located just west of the Secaucus Junction station) and CP Sparrow just west of Port Jervis NY. The tracks run through Mahwah so being here a Ramapo is seemed appropriate.

When it came time to decide how I was going to build this project, I gave myself a few criteria I wanted to meet. The first was I wanted to build it where the drawing the UI wouldn't be extremely difficult; the second was that ideally, I would like it to be cross platform, since there seems to be a ever growing divide between Windows and Mac. To check off both these boxes, I landed on a web application. Allowing it to work on any platform, and it will give me some experience with web development, which is a massive part of the industry. Once I decided on making the application as a web app, I had to choose a framework to create the app in. I first looked into Django which would be written in Python, but Django is more geared towards displaying data, which I great for most web applicants but since what I wanted to create is more like a game, it became clear the ReactJS would be a better choice, because it'll be easier to follow the "Model and View" idea for creating a game, similar to what is taught is the OPL class here.

ReactJS uses Javascript as the main language which was a I was somewhat familiar with after writing the extra credit project for OPL in Javascript for the scripting language project, although that game was created with just basic Javascript and HTML, instead of using a framework like ReactJS. I found that using one of these frameworks will likely help me not only in building the application, but in job interviews a well, since I noticed that all of the internship interviews, I went on asked me if I had any experience with any of these frameworks.

Part II: Installation Instructions

Option 1: There are two options for installing this simulator and running it. The first one is that I installed the application on an Amazon Web Service bucket, so you can just go to a link and it will ack like any other website game.

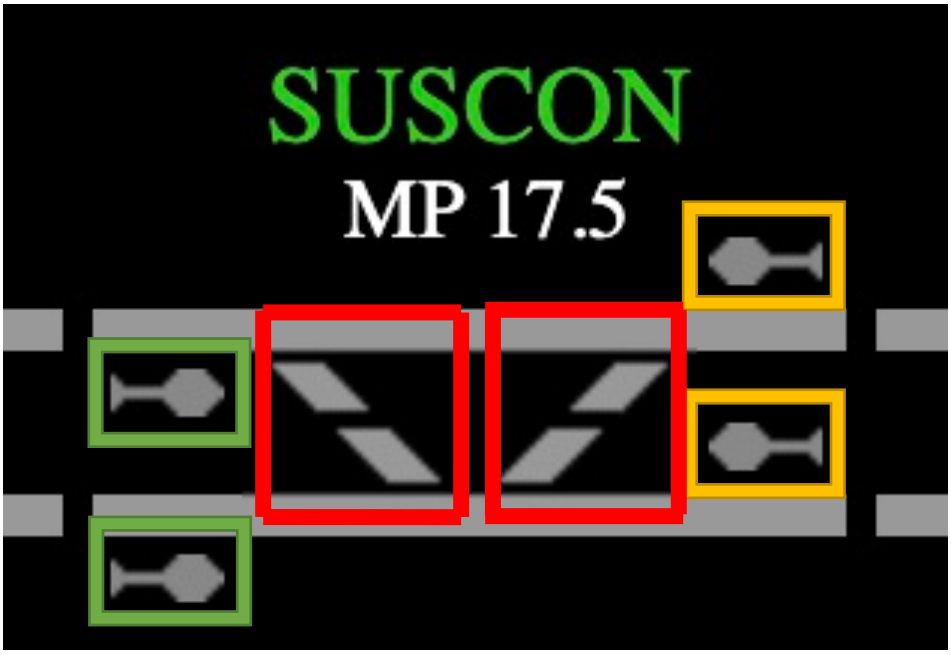
Link:

Option 2: You can install the ReactJS development server which will allow you to run the application in your browser at "Localhost:8080"

- Install the nodeJS package from their website (https://nodejs.org/en/) make sure to select the correct version for you system.
- Once installed open Terminal (or Command Prompt) and cd into the directory that holds the program that you copied off the USB stick provided.
- Once there run "npm start"
- Then everything should be running

Part III: User's Manual

This application works in the same way that program that is used by New Jersey Transit to actually control their railroad works, so if you can use this simulator then you'd be able to walk into the Operations Center in Kearny, NJ and dispatch the real railroad. The first thing you need to know is that going to the left is West and going to the right is East. All of the controls are in each interlocking, basically all you can control is which way the switches are facing, and which signal is lined, see the picture below for more information on how to dispatch.



What is in the Green and Yellow boxes are signals, they represent what is in the real world, that train crews used to know what there route is, think of them similar to traffic lights on the road, although these are not controlled by a timer, but my an actually person.

- The signals in the green boxes are the eastbound signals, these are used by trains that are heading east, so when you click one of these signals, it will line to the next interlocking to the east
- The signals in the yellow boxes are the westbound signals, work the same as the eastbound signals but instead for movement west.

What is in the red boxes are the switches in the interlocking, clicking on these will change the state of the switch, either if it is reversed or normal, the drawing will change on the screen depending on what state the switch is at.

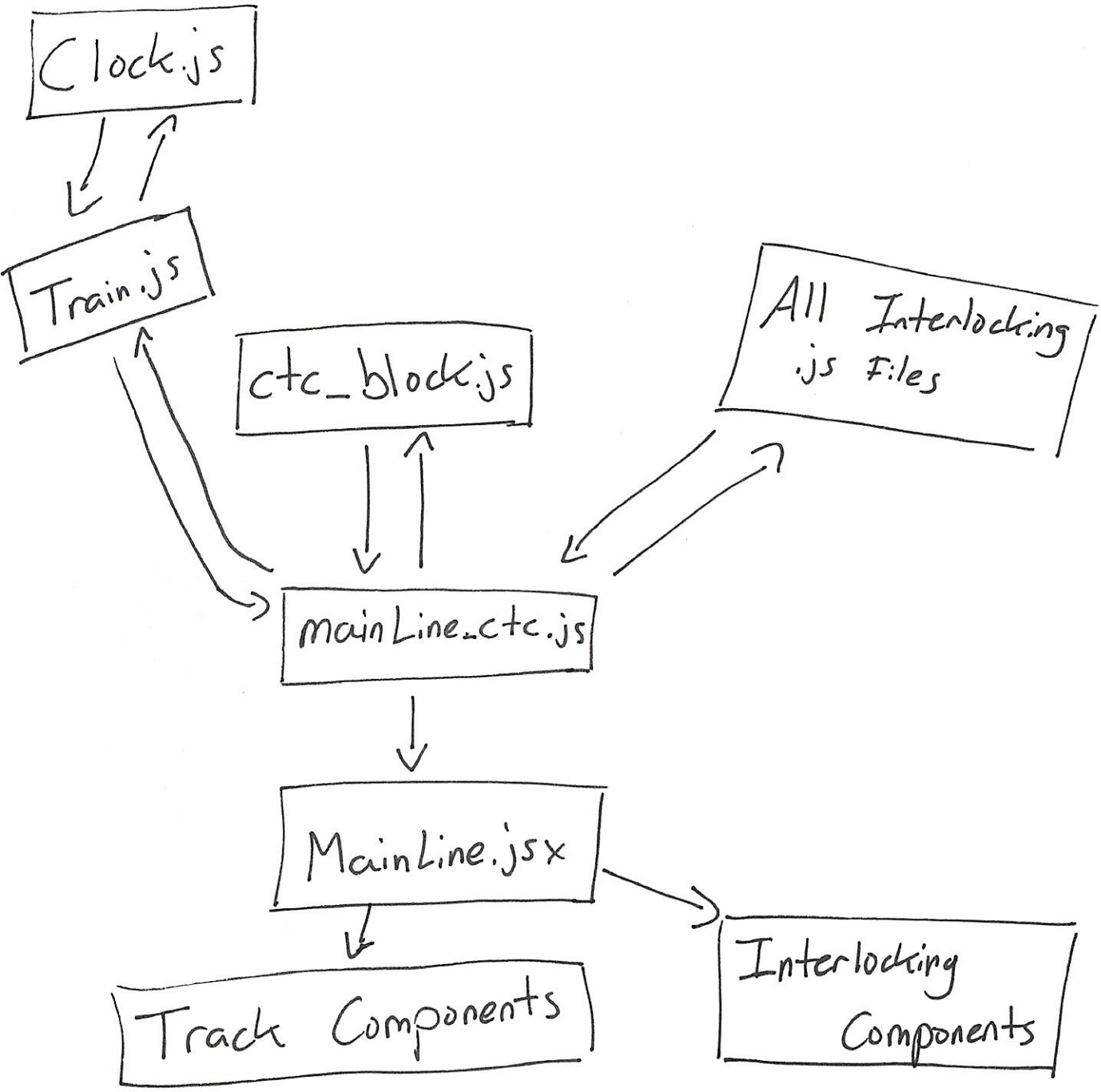
Above is what a reversed switch looks like, it is clear to see that it actually draws which way the train will move when it goes over this section of track.

When the signal is clicked, if there are conflicts with the route you have set up, you will get an alert message. If there are no conflicts then the track will turn green, to denote that there is a route setup between the two interlockings. Also if the track is Red that means that block or piece of track is currently occupied by a train, and the red text above the occupied track is the symbol of that train.

Part IV: Design

For this project, I followed the standard design which is called a “Model and View”, meaning that there are two major parts to the program. The Model holds all the information and is actually running is what the program is doing. And the View is what draws what is going on in the program for the user to see, the View literally lets you view the information. In this design standard, the view cannot change anything in the Model, it can only reference the data from the model. Do build the model, I created a Javascript class for each interlocking, and a main controller class. For the view part is a ReactJS component for each interlocking and tracks on the panel. Using the properties feacture of ReactJS components allows us to pass information to the component, but doesn’t allow it to pass data backwards, instead you have to pass a function to the component, and this function is from the controller class, so when the user clicks something, the corresponding function in the Model class is called.

DATA FLOW



The JSX component only recieve data they donot creat any data themselfs

Part V: File Structure

/Project
- /dist
- /doc
- /node_modules
- /public
  - /images
- /src
  - /components
    - /Panel
      - /Bergen_County_Line
      - /Main_Line
      - /Southern_Tier_Line
  - /css
    - /Bergen_County_Line
    - /Main_Line
    - /Southern_Tier_Line
  - /other
  - /scripts
    - /CTC
    - /Interlockings
      - /Bergen_Line
      - /Main_Line
      - /Southern_Tier_Line
    - /Trains

/Project – directory is what holds the entire project, all of the resource images, and code

/dist – Holds the main HTML file, which actually doesn't have any information in it other than reference to the ReactJS app bundle

/doc – Holds the documentation for the application, where you found this file

/node_modules – This is created by the ReactJS application which is NOT written by me

/public – Holds resource files in this case all it holds is a directory holding all the /images that might be used by the application

/src – Holds all the source code for the application

/src/components – Holds all the JSX files which are the ReactJS components that make up the panel on the screen, they are divided into the sections of the railroad

/src/css – The style sheets for the ReactJS components, these are also divided up by the sections of the railroad they're in

/src/other – Was used by me to hold the work log while coding this application

/src/scripts – Holds all the Javascript "model" files which control the train movements

/src/scripts/CTC – The ctc controller files to control the entire railroad

/src/scripts/Interlockings – All the model classes for each of the interlocking components

/src/scripts/Trains – the classes for the trains that move across the railroad.

Part VI: Class Descriptions

SEE NEXT PAGE

Class: MainLine_CTC

MainLine_CTC()

new MainLine_CTC()

The constructor for the Clock class

constructor()

Source: mainLine_ctc.js, line 84

Methods

add_train()

Takes in a new train and adds it to the train_list array

add_train()

Source: mainLine_ctc.js, line 600

get_bc()

Gets reference to the CP BC Interlocking

get_bc()

Source: mainLine_ctc.js, line 393

Returns:

Reference to the CP BC Interlocking

get_bergen_blocks_status()

Gets the status of all the blocks on the Southern Tier Section

get_bergen_blocks_status()

Source: mainLine_ctc.js, line 772

Returns:

An object with the status of each block

get_bergen_symbols()

Gets all the symbols for the blocks on the Bergen County Line Section

get_bergen_symbols()

Source: mainLine_ctc.js, line 850

Returns:

An obnject with all the block symbols on the Bergen Line

get_block_by_name(name,)

takes in the name of a block, and returns the reference to that specific block

get_block_by_name()

Parameters:

Name	Type	Description
name,		the name of the block to find

Source: mainLine_ctc.js, line 1191

Returns:

reference to the block

get_bt()

Gets reference to the BT Interlocking

get_bt()

Source: mainLine_ctc.js, line 569

Returns:

Reference to the BT Interlocking

get_hall()

Gets reference to the CP Hall Interlocking

get_hall()

Source: mainLine_ctc.js, line 426

Returns:

Reference to the CP Hall Interlocking

get_harriman()

Gets reference to the CP Harriman Interlocking

get_harriman()

Source: mainLine_ctc.js, line 459

Returns:

Reference to the CP Harriman Interlocking

get_hilburn()

Gets reference to the Hilburn Interlocking

get_hilburn()

Source: mainLine_ctc.js, line 481

Returns:

Reference to the Hilburn Interlocking

get_howells()

Gets reference to the CP Howells Interlocking

get_howells()

Source: mainLine_ctc.js, line 415

Returns:

Reference to the CP Howells Interlocking

get_hudson()

Gets reference to the CP Hudson Junction Interlocking

get_hudson()

Source: mainLine_ctc.js, line 437

Returns:

Reference to the CP Hudson Junction Interlocking

get_hx()

Gets reference to the HX Interlocking

get_hx()

Source: mainLine_ctc.js, line 591

Returns:

Reference to the HX Interlocking

get_interlocking_route(key,, direction,)

Takes where a train currently is and gets it's next route

get_interlocking_route()

Parameters:

Name	Type	Description
key,		Is ueed to find the trains curent interlocking
direction,		which way the train is traveling

Source: mainLine_ctc.js, line 952

get_laurel()

Gets reference to the Laurel Interlocking

get_laurel()

Source: mainLine_ctc.js, line 558

Returns:

Reference to the Laurel Interlocking

get_mainLine_blocks_status()

Gets the status of all the bloccks on the Southern Tier Section

get_mainLine_blocks_status()

Source: mainLine_ctc.js, line 725

Returns:

An object with the status of each block

get_mainLine_symbols()

Gets all the symbols for the blocks on the Main Line Section

get_mainLine_symbol()

> Source: mainLine_ctc.js, line 875

Returns:

An object with all the block symbols on the Main Line Section

get_mill()

Gets reference to the Mill Interlocking

get_mill()

> Source: mainLine_ctc.js, line 536

Returns:

Reference to the Mill Interlocking

get_ov()

Gets reference to the CP OV Interlocking

get_ov()

> Source: mainLine_ctc.js, line 404

Returns:

Reference to the CP OV Interlocking

get_pa()

Gets reference to the CP PA Interlocking

get_pa()

> Source: mainLine_ctc.js, line 371

Returns:

Reference to the CP PA Interlocking

get_pascack()

Gets reference to the Pascack Interlocking

get_pascack()

> Source: mainLine_ctc.js, line 580

Returns:

Reference to the Pascack Interlocking

get_port()

Gets reference to the CP Port Interlocking

get_port()

Source: mainLine_ctc.js, line 382

Returns:

Reference to the CP Port Interlocking

get_ridgewood()

Gets reference to the Ridgewood Junction Interlocking

get_ridgewood()

Source: mainLine_ctc.js, line 514

Returns:

Reference to the Ridgewood Junction Interlocking

get_sf()

Gets reference to the SF Interlocking

get_sf()

Source: mainLine_ctc.js, line 492

Returns:

Reference to the SF Interlocking

get_sparrow()

Gets reference to the CP Sparrow Interlocking

get_sparrow()

Source: mainLine_ctc.js, line 360

Returns:

Reference to the CP Sparrow Interlocking

get_sterling()

Gets reference to the CP Sterling Interlocking

get_sterling()

Source: mainLine_ctc.js, line 470

Returns:

Reference to the CP Sterling Interlocking

get_suscon()

Gets reference to the Suscon Interlocking

get_suscon()

Source: mainLine_ctc.js, line 525

Returns:

Reference to the Suscon Interlocking

get_tier_block_status()

get_tier_block_status()

Source: mainLine_ctc.js, line 801

Returns:

An object with the status of each block

get_tier_symbols()

Gets all the symbols for the blocks on the Southern Tier Section

get_tier_symbols()

Source: mainLine_ctc.js, line 914

Returns:

An object with all the block symbols on the Southern Tier Section

get_valley()

Gets reference to the CP Central Valley Interlocking

get_valley()

Source: mainLine_ctc.js, line 448

Returns:

Reference to the CP Central Valley Interlocking

get_wc()

Gets reference to the WC Interlocking

get_wc()

Source: mainLine_ctc.js, line 503

Returns:

Reference to the WC Interlocking

get_westSecaucus()

Gets reference to the West Secaucus Interlocking

get_westSecaucus()

Source: mainLine_ctc.js, line 547

Returns:

Reference to the West Secaucus Interlocking

occupy_blocks()

goes through all the trains and finds their current location and occupys the correct block

occupy_blocks()

Source: mainLine_ctc.js, line 610

reset_route_mainLine_blocks()

Resets all the blocks that are routed

reset_route_mainLine_blocks()

Source: mainLine_ctc.js, line 629

set_occupy_interlocking(track,, name,)

Takes in what interlocking and the track number, and set that the specific interlocking is occupied on the last track

set_occupy_interlocking

Parameters:

Name	Type	Description
track,		the track number in the interlocking to occupy, for some interlocking with only one route doesn't need the track
name,		the name of the interlocking to occupy

Source: mainLine_ctc.js, line 1048

update_interlockings()

Goes through to see if each interlocking can have a train clear if it's occupied

update_interlockings()

Source: mainLine_ctc.js, line 323

update_route_blocks()

Gets all the routes from each interlocking and sets the accoriding blocks

update_route_blocks()

Source: mainLine_ctc.js, line 210

update_trains()

Goes through all the trains in the list and updates their location if they're capable of doing so

updates_trains()

Source: mainLine_ctc.js, line 261

Class: CTC_Block

CTC_Block(p_name,, p_size,, p_status,)

new CTC_Block(p_name,, p_size,, p_status,)

The Constructor of the CTC_Block Class

Sets all the memeber variables to their initial values, when the application starts

Parameters:

Name	Type	Description
p_name,		The Name of the Block
p_size,		The Size of the Block
p_status,		Current Status. Only Used for debugging when build the applications

Source: ctc_block.js, line 36

Methods

get_block_status()

Getter for the block_status member variable

get_block_status()

Source: ctc_block.js, line 50

Returns:

The current status of the block

get_size()

Getter for the block_size member variable

get_size()

Source: ctc_block.js, line 61

Returns:

The size of the block

get_symbol()

Getter for the train_symbol memebr variable

get_symbol()

Source: ctc_block.js, line 72

Returns:

The symbol of the trail that is currently in the block

reset_block()

Resets the Block status to Empty

This is used to reset the block, when the CTC controller refreshes the train and route locations

Source: ctc_block.js, line 83

set_block_status(p_status,)

Sets the block current status based off of what tag is passed in

set_block_status()

Parameters:

Name	Type	Description
p_status,		A String which is the Kinda of status of what to set the block too

Source: ctc_block.js, line 108

set_symbol(n_symbol,)

Setter for the train_symbol member variable

set_symbol()

Parameters:

Name	Type	Description
n_symbol,		The new symbols to set the member variable too

Source: ctc_block.js, line 97

Class: Train

Train(p_symbol, p_location, p_direction, p_block_size)

CLASS Train

Constructor

new Train(p_symbol, p_location, p_direction, p_block_size)

constructor()

Parameters:

Name	Type	Description
p_symbol		> The Train's Symbol
p_location		> The Trains Inital Location
p_direction		> The Direction the train is traveling
p_block_size		> The size of the trains inital block

Source: train.js, line 33

Methods

can_update_location()

can_update_location()

Source: train.js, line 72

get_block_size()

get_block_size()

Source: train.js, line 99

get_direction()

get_direction()

Source: train.js, line 118

get_location()

get_location()

Source: train.js, line 90

get_route()

get_route()

Source: train.js, line 127

get_symbol()

get_symbol()

Source: train.js, line 53

Returns:

The train symbol

set_block_size(n_size,)

set_block_size()

Parameters:

Name	Type	Description
n_size,		the new size of the next block

Source: train.js, line 109

set_route(n_route,)

set_route()

Parameters:

Name	Type	Description
n_route,		the trains new route

Source: train.js, line 137

update_location()

update_location()

Source: train.js, line 62

Class: Clock

Home

Classes

Clock

Clock()

CLASS Clock

Constructor

new Clock()

constructor()

Source: clock.js, line 26

Members

getTimeFromStart

getTimeFromStart()

Source: clock.js, line 46

Methods

startClock()

startClock()

Source: clock.js, line 35

Class: CTC_Hilburn

CTC_Hilburn()

`new CTC_Hilburn()`

The constructor for the CTC_Hilburn class

This will initialize all the member variables when the program is started

Source: ctc_hilburn.js, line 40

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_hilburn.js, line 206

click_sig_2e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hilburn.js, line 149

click_sig_2w_1(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_hilburn.js, line 88

click_sig_2w_2(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

> Source: ctc_hilburn.js, line 118

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

> Source: ctc_hilburn.js, line 265

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

> Source: ctc_hilburn.js, line 246

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

> Source: ctc_hilburn.js, line 64

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hilburn.js, line 188

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_hilburn.js, line 230

Class: CTC_Laurel

CTC_Laurel()

new CTC_Laurel()

The constructor for the CTC_Laurel class

This will initialize all the member variables when the program is started

Source: ctc_laurel.js, line 66

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_laurel.js, line 785

click_sig_2w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_laurel.js, line 158

click_sig_4e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Name	Type	Description
next_block_4,		The next block on Track #4

Source: ctc_laurel.js, line 599

click_sig_4w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_laurel.js, line 226

click_sig_6e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_laurel.js, line 436

click_sig_8e(next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_4,		The next block on Track #4

Source: ctc_laurel.js, line 679

click_sig_8w(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

> Source: ctc_laurel.js, line 293

click_sig_10w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

> Source: ctc_laurel.js, line 372

click_sig_12e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

> Source: ctc_laurel.js, line 516

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_laurel.js, line 962

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_laurel.js, line 845

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_laurel.js, line 115

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_laurel.js, line 709

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_laurel.js, line 728

set_trk_3_occupied(n_state,)

Sets track #3 as occupied

set_trk_3_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_laurel.js, line 747

set_trk_4_occupied(n_state,)

Sets track #4 as occupied

set_trk_4_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_laurel.js, line 766

throw_sw_1()

Function to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 863

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 879

throw_sw_7()

Funtion to throw switch #7 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 895

throw_sw_9()

Funtion to throw switch #9 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 911

throw_sw_11()

Funtion to throw switch #11 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 927

throw_sw_13()

Funtion to throw switch #13 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_laurel.js, line 943

Documentation generated by JSDoc 3.6.3 on Wed Sep 25 2019 00:09:45 GMT-0400 (Eastern Daylight Time)

Class: CTC_Mill

CTC_Mill()

new CTC_Mill()

The constructor for the CTC_Mill class

This will initialize all the member variables when the program is started

Source: ctc_mill.js, line 49

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_mill.js, line 284

click_sig(sigNum,, next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
sigNum,		The number of the signal clicked
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_mill.js, line 85

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_mill.js, line 393

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_mill.js, line 323

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_mill.js, line 336

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_mill.js, line 246

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_mill.js, line 265

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_mill.js, line 360

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

> Source: ctc_mill.js, line 374

Class: CTC_Ridgewood

CTC_Ridgewood()

new CTC_Ridgewood()

The constructor for the CTC_Ridgewood class

This will initialize all the member variables when the program is started

Source: ctc_ridgewood.js, line 59

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_ridgewood.js, line 751

click_sig_2e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_ridgewood.js, line 421

click_sig_2w1(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Name	Type	Description
next_block_3,		The next block on Track #3

Source: ctc_ridgewood.js, line 142

click_sig_2w2(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_ridgewood.js, line 211

click_sig_4e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_ridgewood.js, line 508

click_sig_4w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_ridgewood.js, line 282

click_sig_6e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_ridgewood.js, line 595

click_sig_6w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_ridgewood.js, line 351

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_ridgewood.js, line 884

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_ridgewood.js, line 676

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_ridgewood.js, line 102

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_ridgewood.js, line 694

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_ridgewood.js, line 713

set_trk_3_occupied(n_state,)

Sets track #3 as occupied

set_trk_3_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_ridgewood.js, line 732

throw_sw_1()

Function to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 801

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 817

throw_sw_5()

Funtion to throw switch #5 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 833

throw_sw_7()

Funtion to throw switch #7 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 849

throw_sw_9()

Funtion to throw switch #9 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_ridgewood.js, line 865

Class: CTC_SF

CTC_SF()

new CTC_SF()

The constructor for the CTC_SF class

This will initialize all the member variables when the program is started

Source: ctc_sf.js, line 50

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_sf.js, line 393

click_sig_2e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sf.js, line 227

click_sig_2w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_sf.js, line 117

click_sig_4e_1(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_sf.js, line 260

click_sig_4e_2(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_sf.js, line 309

click_sig_4w(next_block_1,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_3,		The next block on Track #3

Source: ctc_sf.js, line 179

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_sf.js, line 483

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_sf.js, line 464

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_sf.js, line 83

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_sf.js, line 355

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_sf.js, line 374

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sf.js, line 432

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sf.js, line 448

Class: CTC_Suscon

CTC_Suscon()

new CTC_Suscon()

The constructor for the CTC_Suscon class

This will initialize all the member variables when the program is started

Source: ctc_suscon.js, line 48

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_suscon.js, line 282

click_sig(sigNum,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
sigNum,		The signal number that was clicked
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_suscon.js, line 83

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_suscon.js, line 393

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_suscon.js, line 320

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_suscon.js, line 336

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_suscon.js, line 244

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_suscon.js, line 263

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_suscon.js, line 360

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

Source: ctc_suscon.js, line 374

Class: CTC_WC

CTC_WC()

new CTC_WC()

The constructor for the CTC_WC class

This will initialize all the member variables when the program is started

Source: ctc_wc.js, line 57

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_wc.js, line 561

click_sig_2e_1(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 329

click_sig_2e_2(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 395

click_sig_2w_1(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 131

click_sig_2w_2(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 197

click_sig_4e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_wc.js, line 461

click_sig_4w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

> Source: ctc_wc.js, line 263

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

> Source: ctc_wc.js, line 684

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

> Source: ctc_wc.js, line 664

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

> Source: ctc_wc.js, line 94

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_wc.js, line 523

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_wc.js, line 542

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_wc.js, line 600

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_wc.js, line 616

throw_sw_5()

Funtion to throw switch #5 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_wc.js, line 632

throw_sw_7()

Funtion to throw switch #7 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_wc.js, line 648

Class: CTC_WestSecaucus

CTC_WestSecaucus()

new CTC_WestSecaucus()

The constructor for the CTC_WestSecaucus class

This will initialize all the member variables when the program is started

Source: ctc_westSecaucus.js, line 43

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_westSecaucus.js, line 242

click_sig(sigNum,, next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
sigNum,		the id of the signal clicked
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_westSecaucus.js, line 74

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_westSecaucus.js, line 343

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_westSecaucus.js, line 266

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_westSecaucus.js, line 282

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_westSecaucus.js, line 224

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_westSecaucus.js, line 308

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_westSecaucus.js, line 324

Class: CTC_BT

CTC_BT()

new CTC_BT()

The constructor for the CTC_BT class

This will initialize all the member variables when the program is started

Source: ctc_bt.js, line 50

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_bt.js, line 442

click_sig_2e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_bt.js, line 268

click_sig_2w1(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_bt.js, line 117

click_sig_2w2(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_bt.js, line 167

click_sig_4e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_bt.js, line 335

click_sig_4w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_bt.js, line 217

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_bt.js, line 544

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_bt.js, line 525

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_bt.js, line 84

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_bt.js, line 398

set_trk_2_occupied(n_state,)

Sets track #1 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_bt.js, line 420

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_bt.js, line 481

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

Source: ctc_bt.js, line 495

throw_sw_5()

Changes the current state of switch #5, used when user clicks the switch

throw_sw_5()

Source: ctc_bt.js, line 509

Documentation generated by JSDoc 3.6.3 on Wed Sep 25 2019 00:07:20 GMT-0400 (Eastern Daylight Time)

Class: CTC_HX

CTC_HX()

new CTC_HX()

The constructor for the CTC_BT class

constructor()

Source: ctc_hx.js, line 50

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

can_clear()

Source: ctc_hx.js, line 475

click_sig_2e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

click_sig_2e()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_hx.js, line 295

click_sig_2w1(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

click_sig_2w1()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hx.js, line 114

click_sig_2w2(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

click_sig_2w2()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hx.js, line 163

click_sig_2w3(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

click_sig_2w3()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hx.js, line 212

click_sig_4e(next_block_1,, next_block_2,, next_block_3,, next_block_4,)

the function that is called when clicking the signal, creates a route

click_sig_4e()

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3
next_block_4,		The next block on Track #4

Source: ctc_hx.js, line 362

click_sig_4w(next_block_2,)

the function that is called when clicking the signal, creates a route

click_sig_4w()

Parameters:

Name	Type	Description
next_block_2,		The next block on Track #2

Source: ctc_hx.js, line 260

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

get_interlocking_status()

> Source: ctc_hx.js, line 574

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

> Source: ctc_hx.js, line 555

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

> Source: ctc_hx.js, line 84

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

> Source: ctc_hx.js, line 437

set_trk_2_occupied(n_state,)

Sets track #1 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hx.js, line 456

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_hx.js, line 511

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

Source: ctc_hx.js, line 525

throw_sw_5()

Changes the current state of switch #5, used when user clicks the switch

throw_sw_5()

Source: ctc_hx.js, line 539

Class: CTC_Pascack

CTC_Pascack()

new CTC_Pascack()

The constructor for the CTC_BT class

This will initialize all the member variables when the program is started

Source: ctc_pascack.js, line 47

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_pascack.js, line 339

click_sig_2e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pascack.js, line 206

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pascack.js, line 108

click_sig_4e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pascack.js, line 255

click_sig_4w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pascack.js, line 157

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_pascack.js, line 424

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_pascack.js, line 405

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_pascack.js, line 78

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_pascack.js, line 301

set_trk_2_occupied(n_state,)

Sets track #1 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_pascack.js, line 320

throw_sw_1()

Changes the current state of switch #1, used when user clicks the switch

throw_sw_1()

Source: ctc_pascack.js, line 375

throw_sw_3()

Changes the current state of switch #3, used when user clicks the switch

throw_sw_3()

Source: ctc_pascack.js, line 389

Class: CTC_BC

CTC_BC()

new CTC_BC()

The constructor for the CTC_BC class

This will initialize all the member variables when the program is started

Source: ctc_bc.js, line 41

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_bc.js, line 207

click_sig_2e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_bc.js, line 131

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_bc.js, line 90

click_sig_4e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_bc.js, line 161

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_bc.js, line 267

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_bc.js, line 248

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_bc.js, line 65

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_bc.js, line 189

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_bc.js, line 232

Class: CTC_Hall

CTC_Hall()

new CTC_Hall()

The constructor for the CTC_Hall class

This will initialize all the member variables when the program is started

Source: ctc_hall.js, line 47

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check both track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_hall.js, line 299

click_sig_2e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hall.js, line 186

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hall.js, line 107

click_sig_4e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_hall.js, line 231

click_sig_4w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hall.js, line 140

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_hall.js, line 371

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_hall.js, line 352

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving

Name	Type	Description
track,		The Track number of the train

Source: ctc_hall.js, line 77

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hall.js, line 261

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hall.js, line 280

throw_sw_1()

Funtion to throw switch #21 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_hall.js, line 336

Class: CTC_Harriman

CTC_Harriman()

`new CTC_Harriman()`

The constructor for the CTC_Harriman class

This will initialize all the member variables when the program is started

Source: ctc_harriman.js, line 44

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_harriman.js, line 261

click_sig_1e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_harriman.js, line 155

click_sig_1w(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_harriman.js, line 100

click_sig_2e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_harriman.js, line 185

click_sig_3e(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_harriman.js, line 215

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_harriman.js, line 338

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_harriman.js, line 319

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving

Name	Type	Description
track,		The Track number of the train

Source: ctc_harriman.js, line 71

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_harriman.js, line 243

throw_sw_21()

Funtion to throw switch #21 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_harriman.js, line 287

throw_sw_32()

Funtion to throw switch #32 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_harriman.js, line 303

Class: CTC_Howells

CTC_Howells()

`new CTC_Howells()`

The constructor for the CTC_Howells class

This will initialize all the member variables when the program is started

Source: ctc_howells.js, line 39

Methods

`can_clear()`

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_howells.js, line 204

`click_sig_2e(next_block_1,)`

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_howells.js, line 128

`click_sig_2es(next_block_1,)`

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_howells.js, line 158

`click_sig_2w(next_block_1,, next_block_2,)`

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so

what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_howells.js, line 87

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_howells.js, line 264

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_howells.js, line 245

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_howells.js, line 62

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_howells.js, line 186

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_howells.js, line 229

Class: CTC_Hudson

CTC_Hudson()

new CTC_Hudson()

The constructor for the CTC_Hudson class

This will initialize all the member variables when the program is started

Source: ctc_hudson.js, line 43

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_hudson.js, line 295

click_sig_2e(next_block_1,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_3,		The next block on Track #3

Source: ctc_hudson.js, line 190

click_sig_2es(next_block_1,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_3,		The next block on Track #3

Source: ctc_hudson.js, line 235

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hudson.js, line 100

click_sig_2ws(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_hudson.js, line 145

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_hudson.js, line 383

Returns:

Object with the status of the interlocking

get_occupied()

get_occupied()

Source: ctc_hudson.js, line 321

Returns:

If the interlocking is occupied or not

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_hudson.js, line 364

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_hudson.js, line 70

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_hudson.js, line 277

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_hudson.js, line 332

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_hudson.js, line 348

Class: CTC_OV

CTC_OV()

new CTC_OV()

The constructor for the CTC_OV class

This will initialize all the member variables when the program is started

Source: ctc_ov.js, line 41

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_ov.js, line 207

click_sig_2e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_ov.js, line 150

click_sig_2w(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_ov.js, line 89

click_sig_2ws(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_ov.js, line 119

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_ov.js, line 267

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_ov.js, line 248

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_ov.js, line 65

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_ov.js, line 189

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

throw_sw_1()

Source: ctc_ov.js, line 232

Class: CTC_PA

CTC_PA()

new CTC_PA()

The constructor for the CTC_PA class

This will initialize all the member variables when the program is started

Source: ctc_pa.js, line 50

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_pa.js, line 398

click_sig_2e(next_block_1,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_3,		The next block on Track #3

Source: ctc_pa.js, line 251

click_sig_2w_1(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pa.js, line 120

click_sig_2w_2(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pa.js, line 169

click_sig_4e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_pa.js, line 301

click_sig_4w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_pa.js, line 218

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_pa.js, line 505

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_pa.js, line 486

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_pa.js, line 81

set_trk_1_occupied(n_state,)

Sets track #1 as occupied

set_trk_1_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_pa.js, line 360

set_trk_2_occupied(n_state,)

Sets track #2 as occupied

set_trk_2_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_pa.js, line 379

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

throw_sw_1()

Source: ctc_pa.js, line 436

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

throw_sw_3()

Source: ctc_pa.js, line 453

throw_sw_5()

Funtion to throw switch #5 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

throw_sw_5()

Source: ctc_pa.js, line 470

Class: CTC_Port

CTC_Port()

new CTC_Port()

The constructor for the CTC_Port class

This will initialize all the member variables when the program is started

Source: ctc_port.js, line 41

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_port.js, line 207

click_sig_2e_1(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_port.js, line 131

click_sig_2e_2(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_port.js, line 161

click_sig_2w(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so

what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_port.js, line 90

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_port.js, line 266

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_port.js, line 247

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_port.js, line 65

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_port.js, line 189

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_port.js, line 231

Class: CTC_Sparrow

CTC_Sparrow()

new CTC_Sparrow()

The constructor for the CTC_Sparrow class

This will initialize all the member variables when the program is started

Source: ctc_sparrow.js, line 41

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_sparrow.js, line 258

click_sig_2e(next_block_1,, next_block_2,, next_block_3,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2
next_block_3,		The next block on Track #3

Source: ctc_sparrow.js, line 187

click_sig_2w_1(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sparrow.js, line 95

click_sig_2w_2(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sparrow.js, line 125

click_sig_2w_3(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sparrow.js, line 155

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_sparrow.js, line 335

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_sparrow.js, line 316

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving

Name	Type	Description
track,		The Track number of the train

Source: ctc_sparrow.js, line 68

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_sparrow.js, line 240

throw_sw_1()

Funtion to throw switch #1 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sparrow.js, line 284

throw_sw_3()

Funtion to throw switch #3 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sparrow.js, line 300

Class: CTC_Sterling

CTC_Sterling()

new CTC_Sterling()

The constructor for the CTC_Sterling class

This will initialize all the member variables when the program is started

Source: ctc_sterling.js, line 40

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_sterling.js, line 206

click_sig_1e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_sterling.js, line 149

click_sig_2w(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sterling.js, line 88

click_sig_2ws(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_sterling.js, line 118

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_sterling.js, line 265

Returns:

Object with the status of the interlocking

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_sterling.js, line 246

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_sterling.js, line 64

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_sterling.js, line 188

throw_sw_21()

Funtion to throw switch #21 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_sterling.js, line 230

Class: CTC_Valley

CTC_Valley()

new CTC_Valley()

The constructor for the CTC_Valley class

This will initialize all the member variables when the program is started

Source: ctc_valley.js, line 40

Methods

can_clear()

Checks if a track could be cleared, meaning a train is no longer in the interlocking

Check the track if a train has been in the interlocking for more then 4 seconds, if so it clears that track

Source: ctc_valley.js, line 206

click_sig_1e(next_block_1,, next_block_2,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1
next_block_2,		The next block on Track #2

Source: ctc_valley.js, line 149

click_sig_1w(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_valley.js, line 88

click_sig_2w(next_block_1,)

the function that is called when clicking the signal, creates a route

When the function is called it will determine if a route can be created, and if so what the route is and sets it based off of the switch status

Parameters:

Name	Type	Description
next_block_1,		The next block on Track #1

Source: ctc_valley.js, line 118

get_interlocking_status()

returns the status of the interlocking that would be needed by the ReactJS Components

All the information that is returned here is what is needed by the ReactJS Component for the interlocking that is need to draw the interlocking to the screen

Source: ctc_valley.js, line 274

Returns:

Object with the status of the interlocking

get_occupied()

Getter for the int_occupied

get_occupied()

Source: ctc_valley.js, line 228

get_routes()

Gets all the routes from the interlocking

get_routes()

Source: ctc_valley.js, line 255

Returns:

An Array holding every route variable from the interlocking

get_train_route(direction,, track,)

Returns the route for the train at a given track

get_train_route()

Parameters:

Name	Type	Description
direction,		The direction the train is moving
track,		The Track number of the train

Source: ctc_valley.js, line 64

set_occupied(n_state,)

Sets the track as occupied

set_occupied()

Parameters:

Name	Type	Description
n_state,		The new state of the track This was used to test, and never removed passing the state as a paramemter, which is not needed anymore

Source: ctc_valley.js, line 188

throw_sw_21()

Funtion to throw switch #21 in the interlocking The function sets the status of the switch, whether it is is the normal possition of reversed, (True = Reversed / False = Normal)

Source: ctc_valley.js, line 239

Class: MainLine

MainLine(props,)

The React JSX Component Class for the entire Maine Line Dispatcher Panel This class is a JSX React Component for the Maine Line Dispatch Panel, this will control all the other components that make up the pannel. This also controls the functions that allow each component to change their respected back end functions.

Constructor

new MainLine(props,)

The Constructor for the MainLine JSX class. All this does is set that state for every thing getting the information fro the CTC controller, the state here is used to send to the child components so they can render the correct information

constructor()

Parameters:

Name	Type	Description
props,		Required as park of ReactJS, but is not used here

Source: MainLine.jsx, line 90

Members

bc_click_sig_2e

The event handler for Signal #2e

bc_click_sig_2e()

Source: MainLine.jsx, line 1148

bc_click_sig_2w

The event handler for Signal #2w

bc_click_sig_2w()

Source: MainLine.jsx, line 1132

bc_click_sig_4e

The event handler for Signal #4e

bc_click_sig_4e()

Source: MainLine.jsx, line 1163

bc_throw_sw_1

The event handler for switch #1

bc_throw_sw_1()

> Source: MainLine.jsx, line 1178

bt_click_sig_2e

Event handler for the signal #2e

bt_click_sig_2e()

> Source: MainLine.jsx, line 795

bt_click_sig_2w1

Event handler for the signal #2w1

bt_click_sig_2w1()

> Source: MainLine.jsx, line 747

bt_click_sig_2w2

Event handler for the signal #2w2

bt_click_sig_2w2()

> Source: MainLine.jsx, line 763

bt_click_sig_4e

Event handler for the signal #4e

bt_click_sig_4e()

> Source: MainLine.jsx, line 812

bt_click_sig_4w

Event handler for the signal #4

bt_click_sig_4w()

> Source: MainLine.jsx, line 779

bt_throw_sw_1

The event handler for switch #1

bt_throw_sw_1()

> Source: MainLine.jsx, line 829

bt_throw_sw_3

The event handler for switch #3

bt_throw_sw_3()

> Source: MainLine.jsx, line 841

bt_throw_sw_5

The event handler for switch #5

bt_throw_sw_5()

> Source: MainLine.jsx, line 853

hall_click_sig_2e

The event handler for Signal #2e

hall_click_sig_2e()

> Source: MainLine.jsx, line 1346

hall_click_sig_2w

The event handler for Signal #2w

hall_click_sig_2w()

> Source: MainLine.jsx, line 1315

hall_click_sig_4e

The event handler for Signal #4e

hall_click_sig_4e()

> Source: MainLine.jsx, line 1362

hall_click_sig_4w

The event handler for Signal #4w

hall_click_sig_4w()

> Source: MainLine.jsx, line 1330

hall_throw_sw_1

The event handler for switch #1

hall_throw_sw_1()

> Source: MainLine.jsx, line 1377

harriman_click_sig_1e

The event handler for Signal #1e

harriman_click_sig_1e()

> Source: MainLine.jsx, line 1561

harriman_click_sig_1w

The event handler for Signal #1w

harriman_click_sig_1w()

> Source: MainLine.jsx, line 1544

harriman_click_sig_2e

The event handler for Signal #2e

harriman_click_sig_2e()

> Source: MainLine.jsx, line 1576

harriman_click_sig_3e

The event handler for Signal #3e

harriman_click_sig_3e()

> Source: MainLine.jsx, line 1591

harriman_throw_sw_21

The event handler for switch #21

harriman_throw_sw_21()

> Source: MainLine.jsx, line 1606

harriman_throw_sw_32

The event handler for switch #32

harriman_throw_sw_32()

> Source: MainLine.jsx, line 1618

hilburn_click_sig_2e

The event handler for Signal #2e

hilburn_click_sig_2e()

> Source: MainLine.jsx, line 1728

hilburn_click_sig_2w_1

The event handler for Signal #2w_1

hilburn_click_sig_2w_1()

> Source: MainLine.jsx, line 1698

hilburn_click_sig_2w_2

The event handler for Signal #2w_2

hilburn_click_sig_2w_2()

> Source: MainLine.jsx, line 1713

hilburn_throw_sw_1

The event handler for switch #1

hilburn_throw_sw_1()

> Source: MainLine.jsx, line 1744

howells_click_sig_2e

The event handler for Signal #2e

howells_click_sig_2e()

> Source: MainLine.jsx, line 1270

howells_click_sig_2es

The event handler for Signal #2es

howells_click_sig_2es()

> Source: MainLine.jsx, line 1285

howells_click_sig_2w

The event handler for Signal #2w

howells_click_sig_2w()

> Source: MainLine.jsx, line 1254

howells_throw_sw_3

The event handler for switch #3

howells_throw_sw_3()

> Source: MainLine.jsx, line 1300

hudson_click_sig_2e

The event handler for Signal #2e

hudson_click_sig_2e()

> Source: MainLine.jsx, line 1424

hudson_click_sig_2es

The event handler for Signal #2es

hudson_click_sig_2es()

> Source: MainLine.jsx, line 1440

hudson_click_sig_2w

The event handler for Signal #2w

hudson_click_sig_2w()

> Source: MainLine.jsx, line 1392

hudson_click_sig_2ws

The event handler for Signal #2ws

hudson_click_sig_2ws()

> Source: MainLine.jsx, line 1408

hudson_throw_sw_1

The event handler for switch #1

hudson_throw_sw_1()

> Source: MainLine.jsx, line 1456

hudson_throw_sw_3

The event handler for switch #3

hudson_throw_sw_3()

> Source: MainLine.jsx, line 1468

hx_click_sig_2e

The event handler for the Signal 2e

hx_click_sig_2e()

> Source: MainLine.jsx, line 582

hx_click_sig_2w1

The event handler for Signal #2w-1

hx_click_sig_2w1()

> Source: MainLine.jsx, line 519

hx_click_sig_2w2

The event handler for the Signal #2w2

hx_click_sig_2w2()

> Source: MainLine.jsx, line 535

hx_click_sig_2w3

The event handler for the Signal #2w3

hx_click_sig_2w3()

> Source: MainLine.jsx, line 551

hx_click_sig_4e

The event handler for the Signal 4e

hx_click_sig_4e()

> Source: MainLine.jsx, line 599

hx_click_sig_4w

The event handler for the Signal #4w

hx_click_sig_4w()

> Source: MainLine.jsx, line 567

hx_throw_sw_1

The event handler for switch #1

hx_throw_sw_1()

> Source: MainLine.jsx, line 617

hx_throw_sw_3

The event handler for switch #3

hx_throw_sw_3()

> Source: MainLine.jsx, line 629

hx_throw_sw_5

The event handler for switch #5

hx_throw_sw_5()

> Source: MainLine.jsx, line 641

laurel_click_sig_2w

The event handler for Signal #2w

laurel_click_sig_2w()

> Source: MainLine.jsx, line 2489

laurel_click_sig_4e

The event handler for Signal #4e

laurel_click_sig_4e()

> Source: MainLine.jsx, line 2594

laurel_click_sig_4w

The event handler for Signal #4w

laurel_click_sig_4w()

> Source: MainLine.jsx, line 2506

laurel_click_sig_6e

The event handler for Signal #6e

laurel_click_sig_6e()

> Source: MainLine.jsx, line 2558

laurel_click_sig_8e

The event handler for Signal #8e

laurel_click_sig_8e()

> Source: MainLine.jsx, line 2612

laurel_click_sig_8w

The event handler for Signal #8w

laurel_click_sig_8w()

> Source: MainLine.jsx, line 2523

laurel_click_sig_10w

The event handler for Signal #10w

laurel_click_sig_10w()

> Source: MainLine.jsx, line 2541

laurel_click_sig_12e

The event handler for Signal #12e

laurel_click_sig_12e()

> Source: MainLine.jsx, line 2576

laurel_throw_sw_1

The event handler for switch #1

laurel_throw_sw_1()

> Source: MainLine.jsx, line 2627

laurel_throw_sw_3

The event handler for switch #3

laurel_throw_sw_3()

> Source: MainLine.jsx, line 2639

laurel_throw_sw_7

The event handler for switch #7

laurel_throw_sw_7()

> Source: MainLine.jsx, line 2651

laurel_throw_sw_11

The event handler for switch #11

laurel_throw_sw_11()

> Source: MainLine.jsx, line 2663

laurel_throw_sw_13

The event handler for switch #13

laurel_throw_sw_13()

> Source: MainLine.jsx, line 2675

mill_click_sig_2e

The event handler for Signal #2e

mill_click_sig_2e()

> Source: MainLine.jsx, line 2316

mill_click_sig_2w

The event handler for Signal #2w

mill_click_sig_2w()

> Source: MainLine.jsx, line 2299

mill_click_sig_4e

The event handler for Signal #4e

mill_click_sig_4e()

> Source: MainLine.jsx, line 2350

mill_click_sig_4w

The event handler for Signal #4w

mill_click_sig_4w()

> Source: MainLine.jsx, line 2333

mill_throw_sw_1

The event handler for switch #1

mill_throw_sw_1()

> Source: MainLine.jsx, line 2367

mill_throw_sw_3

The event handler for switch #3

mill_throw_sw_3()

> Source: MainLine.jsx, line 2379

ov_click_sig_2e

The event handler for Signal #2e

ov_click_sig_2e()

> Source: MainLine.jsx, line 1223

ov_click_sig_2w

The event handler for Signal #2w

ov_click_sig_2w()

> Source: MainLine.jsx, line 1193

ov_click_sig_2ws

The event handler for Signal #2ws

ov_click_sig_2ws()

> Source: MainLine.jsx, line 1208

ov_throw_sw_1

The event handler for switch #1

ov_throw_sw_1()

> Source: MainLine.jsx, line 1239

pa_click_sig_2e

The event handler for Signal #2e

pa_click_sig_2e()

> Source: MainLine.jsx, line 1011

pa_click_sig_2w_1

The event handler for Signal #2w_1

pa_click_sig_2w_1()

> Source: MainLine.jsx, line 961

pa_click_sig_2w_2

The event handler for Signal #2w_2

pa_click_sig_2w_2()

> Source: MainLine.jsx, line 978

pa_click_sig_4e

The event handler for Signal #4e

pa_click_sig_4e()

> Source: MainLine.jsx, line 1027

pa_click_sig_4w

The event handler for Signal #4w

pa_click_sig_4w()

> Source: MainLine.jsx, line 995

pa_throw_sw_1

The event handler for switch #1

pa_throw_sw_1()

> Source: MainLine.jsx, line 1044

pa_throw_sw_3

The event handler for switch #3

pa_throw_sw_3()

> Source: MainLine.jsx, line 1056

pascack_click_sig_2e

Event handler for the signal #2e

pascack_click_sig_2e()

> Source: MainLine.jsx, line 688

pascack_click_sig_2w

Event handler for the signal #2w

pascack_click_sig_2w()

> Source: MainLine.jsx, line 656

pascack_click_sig_4e

Event handler for the signal #4e

pascack_click_sig_4e()

> Source: MainLine.jsx, line 704

pascack_click_sig_4w

Event handler for the signal #4w

pascack_click_sig_4w()

> Source: MainLine.jsx, line 672

pascack_throw_sw_1

The event handler for switch #1

pascack_throw_sw_1()

> Source: MainLine.jsx, line 720

pascack_throw_sw_3

The event handler for switch #3

pascack_throw_sw_3()

> Source: MainLine.jsx, line 732

port_click_sig_2e_1

The event handler for Signal #2e_1

pa_click_sig_2e_1()

> Source: MainLine.jsx, line 1087

port_click_sig_2e_2

The event handler for Signal #2e_2

pa_click_sig_2e_2()

> Source: MainLine.jsx, line 1102

port_click_sig_2w

The event handler for Signal #2w

pa_click_sig_2w()

> Source: MainLine.jsx, line 1071

port_throw_sw_1

The event handler for switch #1

port_throw_sw_1()

> Source: MainLine.jsx, line 1117

ridgewood_click_sig_2e

The event handler for Signal #2e

ridgewood_click_sig_2e()

> Source: MainLine.jsx, line 2087

ridgewood_click_sig_2w_1

The event handler for Signal #2w_1

ridgewood_click_sig_2w_1()

> Source: MainLine.jsx, line 2019

ridgewood_click_sig_2w_2

The event handler for Signal #2w_2

ridgewood_click_sig_2w_2()

> Source: MainLine.jsx, line 2036

ridgewood_click_sig_4e

The event handler for Signal #4e

ridgewood_click_sig_4e()

> Source: MainLine.jsx, line 2105

ridgewood_click_sig_4w

The event handler for Signal #4w

ridgewood_click_sig_4w()

> Source: MainLine.jsx, line 2053

ridgewood_click_sig_6e

The event handler for Signal #6e

ridgewood_click_sig_6e()

> Source: MainLine.jsx, line 2123

ridgewood_click_sig_6w

The event handler for Signal #6w

ridgewood_click_sig_6w()

> Source: MainLine.jsx, line 2070

ridgewood_throw_sw_1

The event handler for switch #1

ridgewood_throw_sw_1()

> Source: MainLine.jsx, line 2141

ridgewood_throw_sw_3

The event handler for switch #3

ridgewood_throw_sw_3()

> Source: MainLine.jsx, line 2153

ridgewood_throw_sw_5

The event handler for switch #5

ridgewood_throw_sw_5()

> Source: MainLine.jsx, line 2165

ridgewood_throw_sw_7

The event handler for switch #7

ridgewood_throw_sw_7()

> Source: MainLine.jsx, line 2177

ridgewood_throw_sw_9

The event handler for switch #9

ridgewood_throw_sw_9()

> Source: MainLine.jsx, line 2189

sf_click_sig_2e

The event handler for Signal #2e

sf_click_sig_2e()

> Source: MainLine.jsx, line 1792

sf_click_sig_2w

The event handler for Signal #2w

sf_click_sig_2w()

> Source: MainLine.jsx, line 1759

sf_click_sig_4e_1

The event handler for Signal #4e_1

sf_click_sig_4e_1()

Source: MainLine.jsx, line 1807

sf_click_sig_4e_2

The event handler for Signal #4e_2

sf_click_sig_4e_2()

Source: MainLine.jsx, line 1823

sf_click_sig_4w

The event handler for Signal #4w

sf_click_sig_4w()

Source: MainLine.jsx, line 1776

sf_throw_sw_1

The event handler for switch #1

sf_throw_sw_1()

Source: MainLine.jsx, line 1839

sf_throw_sw_3

The event handler for switch #3

sf_throw_sw_3()

Source: MainLine.jsx, line 1851

sparrow_click_sig_2e

The event handler for Signal #2e

sparrow_click_sig_2e()

Source: MainLine.jsx, line 917

sparrow_click_sig_2w_1

The event handler for Signal #2w_1

sparrow_click_sig_2w_1()

Source: MainLine.jsx, line 872

sparrow_click_sig_2w_2

The event handler for Signal #2w_2

sparrow_click_sig_2w_2()

Source: MainLine.jsx, line 887

sparrow_click_sig_2w_3

The event handler for Signal #2w_3

sparrow_click_sig_2w_3()

> Source: MainLine.jsx, line 902

sparrow_throw_sw_1

The event handler for switch #1

sparrow_throw_sw_1()

> Source: MainLine.jsx, line 934

sparrow_throw_sw_3

The event handler for switch #3

sparrow_throw_sw_3()

> Source: MainLine.jsx, line 946

state

Object that holds the state or status information for the component This object holds all the information for everything on the pannel that is required to display the routes correctly

State

> Source: MainLine.jsx, line 99

sterling_click_sig_1e

The event handler for Signal #1e

sterling_click_sig_1e()

> Source: MainLine.jsx, line 1663

sterling_click_sig_2w

The event handler for Signal #2w

sterling_click_sig_2w()

> Source: MainLine.jsx, line 1633

sterling_click_sig_2ws

The event handler for Signal #2ws

sterling_click_sig_2ws()

> Source: MainLine.jsx, line 1648

sterling_throw_sw_21

The event handler for switch #21

sterling_throw_sw_21()

> Source: MainLine.jsx, line 1679

suscon_click_sig_2e

The event handler for Signal #2e

suscon_click_sig_2e()

> Source: MainLine.jsx, line 2221

suscon_click_sig_2w

The event handler for Signal #2w

suscon_click_sig_2w()

> Source: MainLine.jsx, line 2204

suscon_click_sig_4e

The event handler for Signal #4e

suscon_click_sig_4e()

> Source: MainLine.jsx, line 2255

suscon_click_sig_4w

The event handler for Signal #4w

suscon_click_sig_4w()

> Source: MainLine.jsx, line 2238

suscon_throw_sw_1

The event handler for switch #1

suscon_throw_sw_1()

> Source: MainLine.jsx, line 2272

suscon_throw_sw_3

The event handler for switch #3

suscon_throw_sw_3()

> Source: MainLine.jsx, line 2284

update_blocks

This function is called every 0.5 Seconds and updates all the tracks blocks

When this function is called it call 2 functions in the CTC controler class. The first one will check find all the routes at each interlocking and set the correct next block to routed, so the route can be displayed on the pannel The second will get all the trains current locations and make those blocks as occupied, to show the correct location of each train on the pannel

> Source: MainLine.jsx, line 150

update_trains

This function is called every 2 Seconds and updates all the Trains locations

When this function is called it will call 2 functions in the CTC controler The first function updates the trains allowing them to move to the next location if the correct time has be spend in their current block The second function updates the interlockings showing if they are occupied or cleared if the correct time has passed

> Source: MainLine.jsx, line 180

valley_click_sig_1e

The event handler for Signal #1e

valley_click_sig_1e()

> Source: MainLine.jsx, line 1513

valley_click_sig_1w

The event handler for Signal #1w

valley_click_sig_1w()

> Source: MainLine.jsx, line 1483

valley_click_sig_2w

The event handler for Signal #2w

valley_click_sig_2w()

> Source: MainLine.jsx, line 1498

valley_throw_sw_21

The event handler for switch #21

valley_throw_sw_21()

> Source: MainLine.jsx, line 1529

wc_click_sig_2e_1

The event handler for Signal #2e_1

wc_click_sig_2e_1()

> Source: MainLine.jsx, line 1917

wc_click_sig_2e_2

The event handler for Signal #2e_2

wc_click_sig_2e_2()

> Source: MainLine.jsx, line 1934

wc_click_sig_2w_1

The event handler for Signal #2w_1

wc_click_sig_2w_1()

> Source: MainLine.jsx, line 1866

wc_click_sig_2w_2

The event handler for Signal #2w_2

wc_click_sig_2w_2()

> Source: MainLine.jsx, line 1883

wc_click_sig_4e

The event handler for Signal #4e

wc_click_sig_4e()

> Source: MainLine.jsx, line 1951

wc_click_sig_4w

The event handler for Signal #4w

wc_click_sig_4w()

> Source: MainLine.jsx, line 1900

wc_throw_sw_1

The event handler for switch #1

wc_throw_sw_1()

> Source: MainLine.jsx, line 1968

wc_throw_sw_3

The event handler for switch #3

wc_throw_sw_3()

> Source: MainLine.jsx, line 1980

wc_throw_sw_5

The event handler for switch #5

wc_throw_sw_5()

Source: MainLine.jsx, line 1992

wc_throw_sw_7

The event handler for switch #7

wc_throw_sw_7()

Source: MainLine.jsx, line 2004

westSecaucus_click_sig_2e

The event handler for Signal #2e

westSecaucus_click_sig_2e()

Source: MainLine.jsx, line 2411

westSecaucus_click_sig_2w

The event handler for Signal #2w

westSecaucus_click_sig_2w()

Source: MainLine.jsx, line 2394

westSecaucus_click_sig_4e

The event handler for Signal #4e

westSecaucus_click_sig_4e()

Source: MainLine.jsx, line 2445

westSecaucus_click_sig_4w

The event handler for Signal #4w

westSecaucus_click_sig_4w()

Source: MainLine.jsx, line 2428

westSecaucus_throw_sw_1

The event handler for switch #1

westSecaucus_throw_sw_1()

Source: MainLine.jsx, line 2462

westSecaucus_throw_sw_3

The event handler for switch #3

westSecaucus_throw_sw_3()

Source: MainLine.jsx, line 2474

Methods

componentDidMount()

ReactJS function that allows you do set the intervals for when certin functions are called

This function sets the intervals for each function that is called repeadely after a amount of time Will call the update_blocks() function every 0.5 Seconds Will call the update_trains() function every 2 Seconds

Source: MainLine.jsx, line 235

componentWillUnmount()

ReactJS function that removes the intervals, this is never called in this program

This function deletes the intervals that are used to update the blocks & trains This is never called in this program

Source: MainLine.jsx, line 250

render()

standard React function that draws all the other interlockings and track components to the screen

This will draw all the components to the screen to assemble the pannel, it also passes all the function and information to each components through their properties or (props)

Source: MainLine.jsx, line 263

Class: Hilburn

Hilburn()

The React JSX Component Class for the Hilburn Interlocking This class is a JSX React Component for the Hilburn Interlocking, this will control all the UI for the component, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Hilburn()

Source: Hilburn.jsx, line 46

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Hilburn.jsx, line 225

state

Object that holds the state or status information for the component

This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

Source: Hilburn.jsx, line 54

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component

The data that is being changed is passed down from the CTC classes in the simulation backend

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Hilburn.jsx, line 80

render()

standard React function that draws the interlocking to the screen

render()

Source: Hilburn.jsx, line 93

reset_drawings()

Function to reset the signal images and track colors

This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Hilburn.jsx, line 245

set_route_drawings()

Sets the drawing for the route through the interlocking

Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Hilburn.jsx, line 129

Class: Laurel

Laurel()

The React JSX Component Class for the Laurel Interlocking This class is a JSX React Component for the Laurel Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Laurel()

Source: Laurel.jsx, line 68

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Laurel.jsx, line 2007

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Laurel.jsx, line 78

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Laurel.jsx, line 131

render()

standard React function that draws the interlocking to the screen

render()

Source: Laurel.jsx, line 157

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

reset_drawings()

Source: Laurel.jsx, line 2079

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not. There are a lot of possible drawings for this interlocking, which is why the function is so long, I'm not sure if there is a quicker or faster way to accomplish what this function does

set_route_drawings()

Source: Laurel.jsx, line 221

Class: MainLineTracks

MainLineTracks()

The React JSX Component Class for the Tracks in the Main Line portion This class is a JSX React Component for the Main Line Tracks, this will control all the UI for the comonent, showing what blocks are occupied by a train

Constructor

new MainLineTracks()

Source: MainLineTracks.jsx, line 24

Members

state

State

Source: MainLineTracks.jsx, line 34

Methods

componentWillReceiveProps(nextProps,)

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: MainLineTracks.jsx, line 106

render()

render()

Source: MainLineTracks.jsx, line 177

Class: Mill

Mill()

The React JSX Component Class for the Mill Interlocking This class is a JSX React Component for the Mill Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Mill()

Source: Mill.jsx, line 68

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Mill.jsx, line 443

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Mill.jsx, line 78

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Mill.jsx, line 111

render()

standard React function that draws the interlocking to the screen

render()

Source: Mill.jsx, line 129

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

reset_drawings()

Source: Mill.jsx, line 475

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not.

set_route_drawings()

Source: Mill.jsx, line 169

Class: RidgewoodJunction

RidgewoodJunction()

The React JSX Component Class for the Ridgewood Junction Interlocking This class is a JSX React Component for the Ridgewood Junction Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new RidgewoodJunction()

Source: RidgewoodJunction.jsx, line 73

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: RidgewoodJunction.jsx, line 1260

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: RidgewoodJunction.jsx, line 83

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: RidgewoodJunction.jsx, line 132

render()

standard React function that draws the interlocking to the screen

render()

Source: RidgewoodJunction.jsx, line 155

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: RidgewoodJunction.jsx, line 1320

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: RidgewoodJunction.jsx, line 207

Class: SF

SF()

The React JSX Component Class for the SF Interlocking This class is a JSX React Component for the SF Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new SF()

Source: SF.jsx, line 60

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: SF.jsx, line 500

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: SF.jsx, line 70

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: SF.jsx, line 105

render()

standard React function that draws the interlocking to the screen

render()

Source: SF.jsx, line 122

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: SF.jsx, line 530

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: SF.jsx, line 163

Class: Suscon

Suscon()

The React JSX Component Class for the Suscon Interlocking This class is a JSX React Component for the Suscon Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Suscon()

Source: Suscon.jsx, line 67

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Suscon.jsx, line 439

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Suscon.jsx, line 77

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Suscon.jsx, line 108

render()

standard React function that draws the interlocking to the screen

render()

Source: Suscon.jsx, line 125

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

reset_drawings()

Source: Suscon.jsx, line 471

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not.

set_route_drawings()

Source: Suscon.jsx, line 169

Class: WC

WC()

The React JSX Component Class for the WC Interlocking This class is a JSX React Component for the WC Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new WC()

Source: WC.jsx, line 80

Members

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: WC.jsx, line 771

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: WC.jsx, line 90

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description

Name	Type	Description
nextProps,		the new data to set the component state too

Source: WC.jsx, line 132

render()

standard React function that draws the interlocking to the screen

render()

Source: WC.jsx, line 151

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: WC.jsx, line 821

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: WC.jsx, line 198

Class: WestSecaucus

WestSecaucus()

The React JSX Component Class for the West Secaucus Interlocking This class is a JSX React Component for the West Secaucus Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new WestSecaucus()

Source: WestSecaucus.jsx, line 54

Members

set_route_drawing

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: WestSecaucus.jsx, line 155

set_switch_img

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: WestSecaucus.jsx, line 353

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: WestSecaucus.jsx, line 64

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: WestSecaucus.jsx, line 94

render()

standard React function that draws the interlocking to the screen

render()

Source: WestSecaucus.jsx, line 117

Class: BergenTracks

BergenTracks()

The React JSX Component Class for the Tracks in the Bergen County Line portion his class is a JSX React Component for the Bergen County Line Tracks, this will control all the UI for the comonent, showing what blocks are occupied by a train

Constructor

new BergenTracks()

Source: BergenTracks.jsx, line 22

Members

state

Object that holds the state or status information for the component This object holds all the information for the tracks that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: BergenTracks.jsx, line 29

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: BergenTracks.jsx, line 68

render()

standard React function that draws the interlocking to the screen

render()

Source: BergenTracks.jsx, line 107

Class: BT

BT()

The React JSX Component Class for the BT Interlocking This class is a JSX React Component for the BT Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new BT()

Source: BT.jsx, line 73

Members

state

Object that holds the state or status information for the component

This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

Source: BT.jsx, line 80

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: BT.jsx, line 115

render()

standard React function that draws the interlocking to the screen

render()

Source: BT.jsx, line 134

reset_drawings()

Function to reset the signal images and track colors

This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: BT.jsx, line 636

set_route_drawings()

Sets the drawing for the route through the interlocking

Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: BT.jsx, line 176

set_switch_images()

Changes image sources for the switches, depending on switch status

This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

Source: BT.jsx, line 596

Class: HX

HX()

The React JSX Component Class for the HX Interlocking This class is a JSX React Component for the HX Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new HX()

Source: HX.jsx, line 58

Members

state

Object that holds the state or status information for the component

This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

Source: HX.jsx, line 66

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component

The data that is being changed is passed down from the CTC classes in the simulation backend

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: HX.jsx, line 105

render()

standard React function that draws the interlocking to the screen

render()

Source: HX.jsx, line 124

reset_drawings()

Function to reset the signal images and track colors

This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: HX.jsx, line 712

set_route_drawings()

Sets the drawing for the route through the interlocking

Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: HX.jsx, line 168

set_switch_images()

Changes image sources for the switches, depending on switch status

This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

Source: HX.jsx, line 672

Class: PascackJct

PascackJct()

The React JSX Component Class for the Pascack Junction Interlocking This class is a JSX React Component for the Pascack Junction Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new PascackJct()

Source: PascackJct.jsx, line 65

Members

state

Object that holds the state or status information for the component

This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

Source: PascackJct.jsx, line 73

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component

The data that is being changed is passed down from the CTC classes in the simulation backend

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: PascackJct.jsx, line 106

render()

standard React function that draws the interlocking to the screen

render()

Source: PascackJct.jsx, line 124

reset_drawings()

Function to reset the signal images and track colors

This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: PascackJct.jsx, line 445

set_route_drawings()

Sets the drawing for the route through the interlocking

Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: PascackJct.jsx, line 163

set_switch_images()

Changes image sources for the switches, depending on switch status

This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

Source: PascackJct.jsx, line 415

Class: BC

BC()

The React JSX Component Class for the BC Interlocking This class is a JSX React Component for the BC Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new BC()

Source: BC.jsx, line 47

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: BC.jsx, line 57

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: BC.jsx, line 83

render()

standard React function that draws the interlocking to the screen

render()

Source: BC.jsx, line 96

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: BC.jsx, line 246

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: BC.jsx, line 132

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: BC.jsx, line 226

Class: CentralValley

CentralValley()

The React JSX Component Class for the Central Valley Interlocking This class is a JSX React Component for the Central Valley Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new CentralValley()

Source: CentralValley.jsx, line 46

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: CentralValley.jsx, line 56

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: CentralValley.jsx, line 81

render()

standard React function that draws the interlocking to the screen

Source: CentralValley.jsx, line 93

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: CentralValley.jsx, line 248

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: CentralValley.jsx, line 135

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

Source: CentralValley.jsx, line 228

Class: Hall

Hall()

The React JSX Component Class for the Hall Interlocking This class is a JSX React Component for the Hall Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Hall()

Source: Hall.jsx, line 53

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Hall.jsx, line 63

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Hall.jsx, line 94

render()

standard React function that draws the interlocking to the screen

render()

Source: Hall.jsx, line 110

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Hall.jsx, line 365

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Hall.jsx, line 148

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Hall.jsx, line 348

Class: Harriman

Harriman()

The React JSX Component Class for the Harriman Interlocking This class is a JSX React Component for the Harriman Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Harriman()

Source: Harriman.jsx, line 53

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Harriman.jsx, line 63

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Harriman.jsx, line 93

render()

standard React function that draws the interlocking to the screen

render()

Source: Harriman.jsx, line 107

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Harriman.jsx, line 316

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Harriman.jsx, line 146

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Harriman.jsx, line 286

Class: HudsonJunction

HudsonJunction()

The React JSX Component Class for the Hudson Junction Interlocking This class is a JSX React Component for the Hudson Junction Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new HudsonJunction()

Source: HudsonJunction.jsx, line 53

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: HudsonJunction.jsx, line 63

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: HudsonJunction.jsx, line 93

render()

standard React function that draws the interlocking to the screen

render()

Source: HudsonJunction.jsx, line 107

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: HudsonJunction.jsx, line 360

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: HudsonJunction.jsx, line 146

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: HudsonJunction.jsx, line 330

Class: OV

OV()

The React JSX Component Class for the OV Interlocking This class is a JSX React Component for the OV Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new OV()

Source: OV.jsx, line 47

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: OV.jsx, line 57

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: OV.jsx, line 83

render()

standard React function that draws the interlocking to the screen

render()

Source: OV.jsx, line 96

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: OV.jsx, line 246

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: OV.jsx, line 132

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: OV.jsx, line 226

Class: PA

PA()

The React JSX Component Class for the PA Interlocking This class is a JSX React Component for the PA Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new PA()

Source: PA.jsx, line 60

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: PA.jsx, line 70

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: PA.jsx, line 105

render()

standard React function that draws the interlocking to the screen

render()

Source: PA.jsx, line 122

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: PA.jsx, line 535

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: PA.jsx, line 163

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: PA.jsx, line 505

Class: Port

Port()

The React JSX Component Class for the PA Interlocking This class is a JSX React Component for the PA Interlocking, this will control all the UI for the comonent, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Port()

Source: Port.jsx, line 46

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Port.jsx, line 56

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Port.jsx, line 82

render()

standard React function that draws the interlocking to the screen

render()

Source: Port.jsx, line 95

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Port.jsx, line 245

set_route_drawing()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Port.jsx, line 131

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Port.jsx, line 225

Class: SouthernTierTracks

SouthernTierTracks()

The React JSX Component Class for the Tracks in the Southern Tier portion This class is a JSX React Component for the Southern Tier Tracks, this will control all the UI for the comonent, showing what blocks are occupied by a train

Constructor

new SouthernTierTracks()

Source: SouthernTierTracks.jsx, line 24

Members

state

Object that holds the state or status information for the component This object holds all the information for the tracks that is required to display the routes correctly

State

Source: SouthernTierTracks.jsx, line 32

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: SouthernTierTracks.jsx, line 103

render()

standard React function that draws the interlocking to the screen

render()

Source: SouthernTierTracks.jsx, line 173

Class: Sparrow

Sparrow()

The React JSX Component Class for the Sparrow Interlocking This class is a JSX React Component for the Sparrow Interlocking, this will control all the UI for the component, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Sparrow()

Source: Sparrow.jsx, line 53

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Sparrow.jsx, line 63

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Sparrow.jsx, line 93

render()

standard React function that draws the interlocking to the screen

render()

Source: Sparrow.jsx, line 106

reset_drawing()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Sparrow.jsx, line 317

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Sparrow.jsx, line 145

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Sparrow.jsx, line 287

Class: Sterling

Sterling()

The React JSX Component Class for the Hilburn Interlocking This class is a JSX React Component for the Hilburn Interlocking, this will control all the UI for the component, and the click events that will pass reference between the backend and the user. This also controls drawing the route drawings to show if a route(s) is setup in the interlocking or if the route is occupied

Constructor

new Sterling()

Source: Sterling.jsx, line 45

Members

state

Object that holds the state or status information for the component This object holds all the information for the interlocking that is required to display the routes correctly Anything that has "this.props." is passed down from the CTC interlocking class

State

Source: Sterling.jsx, line 55

Methods

componentWillReceiveProps(nextProps,)

Function that updates the state of the component The data that is being changed is passed down from the CTC classes in the simulation backend

componentWillReceiveProps()

Parameters:

Name	Type	Description
nextProps,		the new data to set the component state too

Source: Sterling.jsx, line 81

render()

standard React function that draws the interlocking to the screen

render()

Source: Sterling.jsx, line 94

reset_drawings()

Function to reset the signal images and track colors This function is need, because if the player was to remove a route, or when the train clears the interlocking nothing will clear the route the is displaying on the screen, even if it's gone in the backend

Source: Sterling.jsx, line 244

set_route_drawings()

Sets the drawing for the route through the interlocking Function takes what routes are currently set in the Interlocking class and displays that route in the UI, the drawing will change depending on if the interlocking is occupied or not

Source: Sterling.jsx, line 130

set_switch_img()

Changes image sources for the switches, depending on switch status This function uses the data passed in through status from the CTC classes and shows if the switches are reversed or not on the screen, by changing the image source files, to the correct .png file respectivly

set_switch_img()

Source: Sterling.jsx, line 224

Part VII: Code

SEE NEXT PAGE

```
/**
 * @file index.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary The main extry point for the program
 */
import React from "react";
import ReactDOM from "react-dom";

import MainLine from './components/Panel/MainLine.jsx';

ReactDOM.render(<div> <MainLine /> </div>,
document.getElementById('app'));
```

Part VIII: Testing Plan

How I choose to test my program what do just use it. I always find that it is easier to find issues when you aren't looking for them. So I would just play my game, and see if the program was working correctly, and if I noticed something wrong I'd just write it down and go back and fix it, then recreating the issue to make sure that my fix did actually correct what every problem I found. I know this isn't the most typical way to debug programs, but considering that this is not a typical program, doing case testing didn't seem to do much for me.

Part IV: Summary & Conclusion

When everything was done, this project allowed me to learn and grown on many fronts, first was just on how people create web applications, and how they work. It gave me a much better understanding how the internet works. And now I find myself thinking about how each website I visit was designed and if I was tasked with creating a similar app, how would I create it. This I hope will be helpful when job hunting, since now I have some serious experience developing a web application, I'll be confident to talk about them on Interviews. Another thing this project taught me was time management. The spring semester was quite busy for me, so although I worked on it, I didn't go deep into the program until the summer. I had to make sure I kept on working on the project, even though It was easy for the time to get away from you. It happened a few times when I said I would take a week off for various reasons, and then in a blink of an eye it was 2 or 3 weeks later.

And the last major thing working on this project taught me is that it's okay to admit that you are going in the wrong direction, and completely start over. We reached the end of the semester, the program I had was nowhere near good enough and had a very poor design. It also wasn't using any sort of framework, which would make it very difficult to upload the project to a webserver. I got to the point where I didn't want to work on the program, because it was so poorly designed. So, I decided to completely scrap what I had and started again. Which was probably the best thing I could have done. Allowing me do start over again made the design flow a lot easier, and the code seemed to naturally write it's self once I had the correct design.

If I had to do this program all over again, I would tell myself to do more work earlier on, to let me do less work at the end of the time working on it. Sleep is good, and towards the end of this project I definitely didn't get enough of it.

Part X: Work Log

SEE NEXT PAGE

March 21, 2019:
    - Finished drawing out the layout of the pannel on paper
    (1.5 Hours)

    - Worked on creating the pannel in HTML & CSS
    (2 Hours)

August 20, 2019:
    - Started work to convert to a ReactJS app, started with tutorials and research.
    (2 Hours)

    - Tried to create first component, which would be a crossover button, which took
    4 hours to get an image in appear in ReactJS
    (4.5 Hours)

August 21, 2019:
    - Changed the componet which was going to be just a crossover button to the entire
    Suscon interlocking. Deciding that each interlocking will be it's own component.
    (3 Hours)

August 23, 2019:
    - Tried to figure out how to change an image in ReactJS, which ended up being a
    real big pain, having to import each image file similar to importing a scrip file.
    This solution seems weird but it's the only way I've been getting this to work so far.
    (4 Hours)

August 24, 2019:
    - Started to get all the crossover buttons working, and changing the image based on clicks,
    did this for both Suscon and Mill
    (1 Hour)

    - Converted what I did for the crossover buttons to use the 'state' object which is part of
    a ReactJS Component to more closely follow the conventions of ReactJS
    (2 Hours)

    -

August 25, 2019:
    - Set up the click function for the signals in Suscon
    (0.5 Hours)

- Changed the setup so all the components that make up the Main Line section are wrapped in one
component, this will make it eaiser for passing information from the components to the CTC controller
both ways.
(0.5 Hours)

- Changed all the track block colors to be based off the state in their component class.
(0.25 Hours)

- Setup the beginnings of the CTC Controller class which will act as the backend to controll all the
train movements, also created a ctc_block class which is be a instance for each track block on the railroad
(0.75 Hours)

- Tested out have the informmation in the CTC Controller class change the state of the MainLine component
by using the props property that is built into react to pass the status of 2 blocks into the component itself.
Luckily it works, and I now have the ability CTC Controller change the pannel.
(0.5 Hours)

August 26, 2019:
- Changed how to pass an object as props in the ReactJS component, instead of a variable for each piece of
information that needs to be passed
(0.5 Hours)

- Created a script for the Suscon interlocking to controll the train movements, and started to make this class
change the state of the UI and have the changes reflected in either the ReactJS component or the JS script
(1.5 Hours)

- Got the ReactJS component to send it's changes up to the script class for the interlocking, and having the
changes in the the script be reflecting in the Component, at this point the data is being sent both ways for
the curent status of the crossovers in the interlocking
(1 Hour)

August 27, 2019:
- Started converting Ridgewood Junction from HTML and CSS to a new ReactJS component, also made some changes from
my original design, and learned you do draw a diagonal line in CSS
(0.75 Hours)

- Converted Laurel to a new ReactJS component and cleaned up some of my previous designs
(0.5 Hours)

- Did some work on my drawing for the switch buttons on the pannel
(0.25 Hours)

- Converted the West Secaucus from into a ReactJS component
(0.5 Hours)

August 28, 2019:
- Drawing the Bergen County Line, finished about 95% of the line, creating all the components.
(5 Hours)

August 29, 2019:
- Finished Drawing all the tracks on the Bergen County side of the pannel
(1 Hours)

- Finished drawing of Laurel
(0.25 Hours)

- Finished drawing of ridgewood Junction
(0.25 Hours)

August 30, 2019:
- Reconfigured pascack Junction
(0.25 Hours)

- Finished the drawing in HX
(0.5 Hours)

- Finished drawings in Pascack Junction
(0.25 Hours)

- Finished drawings in BT and add tags for tracks that lead to other lines
(0.25 Hours)

- Orgainized the files structure a bit, and dealt with all the changing of
import statments
(0.25 Hours)

- Created a component for Harriman, starting the next line of tracks for the pannel
(0.5 Hours)

- Created a comonent for Sterling, and the tracks between the interlockings, also works on
parts of hilburn, and sf for the leads to the yard between the hilburn and sf
(1 Hour)

August 31, 2019:
- Finished drawing the the Main Line
(2 Hours)

- Continuted work on the Southern Tier line, finished about 85%
(4.5 Hours)

September 1, 2019:
- Finished All drawings on the pannel
(2 Hours)

September 2, 2019:
- Started connecting all the track blocks on the Main Line to the ctc scripts, now having
the ctc class controlling the front end of the pannel.
(2.5 Hours)

September 3, 2019:
- Fixed the bugs I was having trying to connect the ctc class to show the correct informating
when the frontend refreshes the screen.
(0.75 Hours)

- Created a new class for a game clock, so start trying to get the trains to move accross the
screen to
(0.5 Hours)

- Debugged a problem in the clock class, and now I have a hardcoded train moving accross some of
the blocks
(0.25 Hours)

- Configured the ctc_suscon class to create a route through the interlocking based on the status
of the switches and what signal you click
(0.5 Hours)

- Got the Suscon component to draw the routing in the UI givien the route status of the ctc_suscon
class
(4 Hours)

- Now the route draw turns to occupied if you set the interlocking

to that state and created a
    function to deal with that and will also clear the drawing when the hard coded train passes the
    location
    (2 Hours)

September 4, 2019:
    - Finally found a good way to store and display routes, using something that I
    already had in place.
    (1 Hour)

    - Created a class for the the trains, and now have a actual train moving around the pannel
    (2.5 Hours)

    - Figures out how to have the routes passes to the train, and now the train can move based off of
    how the player has created routes on the pannel
    (3 Hours)

    - Spent a while trying to get the interlocking to become occupied when a train is present, have yet
    to find a good way to do this and will have to revist it.
    (1.5 Hours)

September 5, 2019:
    - Again tried to figure out how to occupie the interlocking in a good way, and still have yet to
    come up with a ellagent solution
    (2 Hours)

    - West Secaucus is now fully operational
    (3 Hours)

    - Routing for Ridgewood Junction is now complete, all thats left is getting the drawings up and
    running, which might by interesting
    (2 Hours)

    - Testing 2 trains at once, and it seems to be working finished
    (0.5 Hours)

September 6, 2019:
    - Reworked how routes are given to trains to make it much easier to have trains to be going in both
    directions
    (1.5 Hours)

    - Fixed issue with the routings for all the current interlockings

(2 Hours)

September 7, 2019:
- Finished all the drawing for routes in Ridgewood, and have decided to finish the routing first, and
then go back to the drawings, because the take the most time
(6 Hours)

September 9, 2019:
- Finsihed the routings for Laurel
(2.5 Hours)

- Finished the routings at the WC interlocking, will still have to go back and add the drawins when
a route is lined
(1.5 Hours)

- Finished the routings in the SF interlocking
(1 Hours)

- Finished the routings in the Hilburn interlocking
(0.5 Hours)

- Did some debugging to fix an minor issue with train movements through hilburn interlocking, at this
point now a train can run from Laurel to Sterling
(0.25 Hours)

September 10, 2019:
- Got the switch in sterling to throw
(0.25 Hours)

- Connected all the blocks on the pannel to actual block classes in the ctc class
(1 Hour)

- Fixed a bug in the suscon interlocking routing
(0.25 Hours)

September 11, 2019:
- Hooked up the blocks on the southern tier section to the ctc mainline class
(0.5 Hours)

- Finished the switches and routing for Sterling, Harriman, and Central Valley interlockings
(2 Hours)

September 12, 2019:
- Finished Switches and routing for Hudson Junction

(0.5 Hours)

- Finished switches and routing for Howells
(0.5 Hours)

- Finished switches and routing for CP OV and for CP BC
(1 Hours)

September 15, 2019:
- Finished Switches and routins for CP Port, CP PA and CP Sparrow. Trains can now run the entire
length of the pannel
(2 Hours)

- Finished route drawings for CP BC
(0.5 Hours)

- Finished route drawings for CP OV
(0.25 Hours)

- Finished route drawings for CP Howells
(0.25 Hours)

September 16, 2019:
- Finished route drawings for CP Port
(0.25 Hours)

- Finished route drawigns fro CP Hall
(0.5 Hours)

- Finished route drawings for CP Sparrow
(0.25 Hours)

- Finished route drawings for CP Hudson Junction
(0.25 Hours)

- Finished route drawings for CP Central Valley
(0.25 Hours)

- Finally figured out how to occupy and interlocking, right now it just working for interlocking that have only one route,
but this is the majority of the interlockings, so I'll be able to get most of the interlockings working 100% in short order.
(0.5 Hours)

- Added occuping interlocking for CP Hudson Junction
(0.5 Hours)

- Commenting
(0.5 Hours)

- Can now occupy CP Hall, this is an interlocking that can have multiple routes, which was gonna be
more difficult than one route interlockings to get it to work.
(0.5 Hours)

- Can now occupied CP Howells
(0.25 Hours)

- Can now occupy CP BC and CP OV and CP Port
(0.5 Hours)

September 17, 2019:
- Drawings and occuping for CP PA
(1.5 Hours)

- Can now occupy CP Sparrow
(0.25 Hours)

- Drawing and occupy at Hilburn
(0.5 Hours)

- Drawings and Occupy at CP Sterling
(0.5 Hours)

- Drawing and Occupy finsihed at CP Harriman
(0.5 Hours)

- General Bug Fixes
(0.5 Hours)

- Drawigns and Occupy finished at SF
(0.75 Hours)

- Occupy finished at West Secaucus
(0.25 Hours)

- Route drawings and Occupy finished at Suscon
(0.75 Hours)

September 18, 2019:
- Route drawings and Occupy finished at Mill
(0.25 Hours)

- Bug Fixes at CP Port
(0.25 Hours)

- Trains now delete if they reach the ends of the railraod, either when they reach a
yard, or the end of pannels west of CP Sparrow or east of Laurel

(0.5 Hours)

- Fixed all east facing yard leads so trains to leave from them, had to change the way
those blocks were named
(0.25 Hours)

- Route drawings and Occupy finished at Ridgewood Junction
(3 Hours)

- Route drawings and Occupy finished at WC
(1.5 Hours)

- Fixing the Blocks on the Bergen County Line
(0.75 Hours)

- Setting up the switches to working at BT
(0.5 Hours)

- Setting up the switches at both Pascack and HX, now all the interlockings have working
switches
(0.75 Hours)

- Routing working for the BT interlocking
(0.5 Hours)

September 19, 2019:
- Routing working at Pascack interlocking
(0.5 Hours)

- Routing working at HX interlocking
(0.5 Hours)

- Bug fixes with routing around the laurel area
(0.75 Hours)

- Route drawings and occupy working at BT interlocking
(1 Hour)

- Route drawings and occupy working at Pascack interlocking
(0.75 Hours)

- Route drawings and occupy working at HX interlocking
(1 Hours)

September 20, 2019:
- Working on Drawings for Laurel Interlocking, finished about a third of them
(2 Hours)

- Debugging for the Laurel Interlocking
(0.75 Hours)

- Continued Drawings for Laurel Interlocking, finsihed about 2/3 of the them at this point
(2 Hours)

September 21, 2019:
- Finished drawings for the laurel interlocking and debugging some of the drawings that change
when another track has a different status.
(2 Hours)

- Finished the routing and occupy for Laurel Interlocking
(0.5 Hours)

- Rewrote the drawings for the WC interlocking using the new system I created and used at Laurel,
this fixed the bugs that had arrised in that interlocking
(3 Hours)

- Started adding symbol tags to the pannel for the blocks, finished all the symbols on the
southern tier section of the pannel
(1.5 Hours)

- Documenting Laurel.jsx
(0.75 Hours)

- Bug fixes at West Secaucus
(0.25 Hours)

September 22, 2019:
- Finished locations on all the symbols for the main line, still have to get the workings.
(0.75 Hours)

- CSS fixes for the text symbos on the pannel to keep it from moving around if the windo is resized
(0.25 Hours)

- Got the symbols of the Main Line working correctly
(1 Hour)

- Got the trains to get the size of the current block so it's more realistically timed as a train
moves accross the railroad.
(1 Hour)

- Finished the symbols on the bergen line
(0.75 Hours)

- Finished commenting for all of Laurel Interlocking
(0.5 Hours)

- Finished commenting for all of Mill Interlocking
(0.5 Hours)

- Finsihed commenting on all of the JSX components for the Main Line section
(2.5 Hours)

- Finsihed commenting on all of the JSX components for the Bergen County Line section
(1 Hour)

September 23, 2019:
- Commenting on JSX components
(3 Hours)

- Commenting finished on all CSS Files
(1 Hour)

- Commenting on all controller JS classes for the Bergen Line & Southern Tier Line
(4 Hours)

- Finish commenting on all the controller JS classes which was for the Main Line and others
(3 Hours)

September 24, 2019:
- Finished the manual and what ever was left of the documentation
(6 Hours)

TOTAL: 156.25 Hours