

```

/**
 * @file ctc_pa.js
 * @author Joey Damico
 * @date September 25, 2019
 * @summary CTC Controller Class for the CP PA Interlocking
 */

// Color Constants For Drawing Routes
const Empty = '#999999';
const Lined = '#75fa4c';
const Occupied = '#eb3323';

/**
 * Class is the Backend for the CP PA Interlocking This class is what
 * controls the CP PA Interlocking,
 * it is sort of like a backen, but is the controller, this is what
 * makes all the train movements possible,
 * and the ReactJS Component class gets information from this class to
 * display the correct status of the
 * interlocking on the screen
 *
 * MEMBER VARIABLES
 * @member sw_1 -> Bool if Switch #1 is Reveresed or Not
 * @member sw_3 -> Bool if Switch #3 is Reveresed or Not
 *
 * @member sig_2w1 -> Bool if Signal #2w-1 is Lined or Not
 * @member sig_2w2 -> Bool if Signal #2w-2 is Lined or Not
 * @member sig_4w -> Bool if Signal #4w is Lined or Not
 * @member sig_2e -> Bool if Signal #2e is Lined or Not
 * @member sig_4e -> Bool if Signal #4e is Lined or Not
 *
 * @member route_w_trk_1 = The west bound route for track #1
 * @member route_w_trk_2 = The west bound route for track #2
 * @member route_w_trk_3 = The west bound route for track #3
 * @member route_e_trk_1 = The east bound route for track #1
 * @member route_e_trk_2 = The east bound route for track #2
 *
 * @member routed_trk_1 = Bool if track #1 is routed or not
 * @member routed_trk_2 = Bool if track #2 is routed or not
 * @member trk_1_time = The time track #1 was occupied, used to know
when to clear the route
 * @member trk_2_time = The time track #2 was occupied, used to know
when to clear the route
 * @member trk_1_occupied = Bool if track #1 is occupied or not
 * @member trk_2_occupied = Bool if track #2 is occupied or not
 */
class CTC_PA {
  /**
   * constructor()

```

```

    * @summary The constructor for the CTC_PA class
    *
    * @description This will initialize all the member variables when
the program is started
    */
    constructor() {
        // Bools for the switches
        this.sw_1 = false;
        this.sw_3 = false;
        // Bools for the signals
        this.sig_2w_1 = false;
        this.sig_2w_2 = false;
        this.sig_4w = false;
        this.sig_2e = false;
        this.sig_4e = false;
        // Track routes
        this.route_w_trk_1 = null;
        this.route_w_trk_2 = null;
        this.route_w_trk_3 = null;
        this.route_e_trk_1 = null;
        this.route_e_trk_2 = null;
        // Used for routing and occupying the tracks
        this.routed_trk_1 = null;
        this.routed_trk_2 = null;
        this.trk_1_time = null;
        this.trk_2_time = null;
    }
    // ---- END constructor() ----

    /**
    * get_train_route()
    * @summary Returns the route for the train at a given track
    *
    * @param direction, The direction the train is moving
    * @param track, The Track number of the train
    */
    get_train_route(direction, track) {
        if (direction === "WEST") {
            if (track === "1") {
                return this.route_w_trk_1;
            }
            else if (track === "2") {
                return this.route_w_trk_2;
            }
            else if (track === "3") {
                return this.route_w_trk_3;
            }
            else {
                return this.route_w_trk_4;
            }
        }
    }

```

```

    }
    else {
        if (track === "1") {
            return this.route_e_trk_1;
        }
        else if (track === "2") {
            return this.route_e_trk_2;
        }
        else {
            return this.route_e_trk_3;
        }
    }
}
// ---- END get_train_route() ----

/**
 * click_sig_2w1()
 * @summary the function that is called when clicking the signal,
creates a route
 *
 * @description 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
 *
 * @param next_block_1, The next block on Track #1
 * @param next_block_2, The next block on Track #2
 */
click_sig_2w_1(next_block_1, next_block_2, next_block_4) {
    if (this.sw_1) {
        return;
    }
    else if (!this.sw_3) {
        if (this.sig_2w_1) {
            this.route_w_trk_1 = null;
            this.routed_trk_1 = false;
            this.sig_2w_1 = false;
        }
        else {
            if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                alert("Cannot Line Route Because Conflict With
Next Block");
                return;
            }
            this.route_w_trk_1 = "W_1_1__|__1_sparrow_pa";
            this.routed_trk_1 = true;
            this.sig_2w_1 = true;
        }
    }
}

```

```

        else if (this.sw_3) {
            if (this.sig_2w_1) {
                this.route_w_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2w_1 = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_w_trk_1 = "W_1_2__|__2_sparrow_pa";
                this.routed_trk_1 = true;
                this.sig_2w_1 = true;
            }
        }
    }
}
// ---- END click_sig_2w1() ----

/**
 * click_sig_2w2()
 * @summary the function that is called when clicking the signal,
creates a route
 *
 * @description 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
 *
 * @param next_block_1, The next block on Track #1
 * @param next_block_2, The next block on Track #2
 */
click_sig_2w_2(next_block_1, next_block_2, next_block_4) {
    if (!this.sw_1) {
        return;
    }
    else if (!this.sw_3) {
        if (this.sig_2w_2) {
            this.route_w_trk_3 = null;
            this.routed_trk_1 = false;
            this.sig_2w_2 = false;
        }
        else {
            if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                alert("Cannot Line Route Because Conflict With
Next Block");
                return;
            }

```

```

        }
        this.route_w_trk_3 = "W_3_1__|__1_sparrow_pa";
        this.routed_trk_1 = true;
        this.sig_2w_2 = true;
    }
}
else if (this.sw_3) {
    if (this.sig_2w_2) {
        this.route_w_trk_3 = null;
        this.routed_trk_1 = false;
        this.sig_2w_2 = false;
    }
    else {
        if (next_block_2 === Occupied || next_block_2 ===
Lined) {
            alert("Cannot Line Route Because Conflict With
Next Block");
            return;
        }
        this.route_w_trk_3 = "W_3_2__|__2_sparrow_pa";
        this.routed_trk_1 = true;
        this.sig_2w_2 = true;
    }
}
}
// ---- END click_sig_2w2() ----

/**
 * click_sig_4w()
 * @summary the function that is called when clicking the signal,
creates a route
 *
 * @description 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
 *
 * @param next_block_1, The next block on Track #1
 * @param next_block_2, The next block on Track #2
 */
click_sig_4w(next_block_2, next_block_4) {
    if (this.sw_3) {
        return;
    }
    else {
        if (this.sig_4w) {
            this.route_w_trk_2 = null;
            this.routed_trk_2 = false;
            this.sig_4w = false;
        }
    }
}

```

```

        else {
            if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                alert("Cannot Line Route Because Conflict With
Next Block");
                return;
            }
            this.route_w_trk_2 = "W_2_2__|__2_sparrow_pa";
            this.routed_trk_2 = true;
            this.sig_4w = true;
        }
    }
}
// ---- END click_sig_4w() ----

/**
 * click_sig_2e()
 * @summary the function that is called when clicking the signal,
creates a route
 *
 * @description 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
 *
 * @param next_block_1, The next block on Track #1
 * @param next_block_3, The next block on Track #3
 */
click_sig_2e(next_block_1, next_block_3) {
    if (this.sw_3) {
        return;
    }
    else if (!this.sw_1) {
        if (this.sig_2e) {
            this.route_e_trk_1 = null;
            this.routed_trk_1 = false;
            this.sig_2e = false;
        }
        else {
            if (next_block_1 === Occupied || next_block_1 ===
Lined) {
                alert("Cannot Line Route Because Conflict With
Next Block");
                return;
            }
            this.route_e_trk_1 = "E_1_1__|__1_pa_port";
            this.routed_trk_1 = true;
            this.sig_2e = true;
        }
    }
}

```

```

        else {
            if (this.sig_2e) {
                this.route_e_trk_1 = null;
                this.routed_trk_1 = false;
                this.sig_2e = false;
            }
            else {
                if (next_block_3 === Occupied || next_block_3 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_1 = "E_1_3__|__0_portYard_west";
                this.routed_trk_1 = true;
                this.sig_2e = true;
            }
        }
    }
    // ---- END click_sig_2e() ----

    /**
     * click_sig_4e()
     * @summary the function that is called when clicking the signal,
creates a route
     *
     * @description 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
     *
     * @param next_block_1, The next block on Track #1
     * @param next_block_2, The next block on Track #2
     * @param next_block_3, The next block on Track #3
    */
    click_sig_4e(next_block_1, next_block_2, next_block_3) {
        if (!this.sw_3) {
            if (this.sig_4e) {
                this.route_e_trk_2 = null;
                this.routed_trk_2 = false;
                this.sig_4e = false;
            }
            else {
                if (next_block_2 === Occupied || next_block_2 ===
Lined) {
                    alert("Cannot Line Route Because Conflict With
Next Block");
                    return;
                }
                this.route_e_trk_2 = "E_2_2__|__2_pa_bc";

```

```

        this.routed_trk_2 = true;
        this.sig_4e = true;
    }
}
else if (this.sw_3 && !this.sw_1) {
    if (this.sig_4e) {
        this.route_e_trk_2 = null;
        this.routed_trk_2 = false;
        this.sig_4e = false;
    }
    else {
        if (next_block_1 === Occupied || next_block_1 ===
Lined) {
            alert("Cannot Line Route Because Conflict With
Next Block");
            return;
        }
        this.route_e_trk_2 = "E_2_1__|__1_pa_port";
        this.routed_trk_2 = true;
        this.sig_4e = true;
    }
}
else if (this.sw_3 && this.sw_1) {
    if (this.sig_4e) {
        this.route_e_trk_2 = null;
        this.routed_trk_2 = false;
        this.sig_4e = false;
    }
    else {
        if (next_block_3 === Occupied || next_block_3 ===
Lined) {
            alert("Cannot Line Route Because Conflict With
Next Block");
            return;
        }
        this.route_e_trk_2 = "E_2_3__|__0_portYard_west";
        this.routed_trk_2 = true;
        this.sig_4e = true;
    }
}
}
// ---- END click_sig_4e() ----

/**
 * set_trk_1_occupied()
 * @summary Sets track #1 as occupied
 *
 * @param 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

```



```

    */
    set_trk_1_occupied(n_state) {
        if (n_state === true) {
            this.trk_1_occupied = n_state;
            this.routed_trk_1 = false;
            this.trk_1_time = new Date().getTime() / 1000;
        }
        else {
            console.log("ERROR");
        }
    }
}
// ---- END set_trk_1_occupied() ----

/**
 * set_trk_2_occupied()
 * @summary Sets track #2 as occupied
 *
 * @param 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
 */
set_trk_2_occupied(n_state) {
    if (n_state === true) {
        this.trk_2_occupied = n_state;
        this.routed_trk_2 = false;
        this.trk_2_time = new Date().getTime() / 1000;
    }
    else {
        console.log("ERROR");
    }
}
// ---- END set_trk_2_occupied() ----

/**
 * can_clear()
 * @summary Checks if a track could be cleared, meaning a train is
no longer in the interlocking
 *
 * @description Check the track if a train has been in the
interlocking for more then 4 seconds, if so it
 * clears that track
 */
can_clear() {
    // Get Current Time
    let current_time = new Date().getTime() / 1000;

    // Track #1
    if (current_time - this.trk_1_time > 4 && current_time -
this.trk_1_time< 100000) {
        this.sig_2w = false;
    }
}

```

```

        this.sig_2e = false;

        this.route_w_trk_1 = null;
        this.route_e_trk_1 = null;
        this.routed_trk_1 = false;

        this.trk_1_occupied = false;
        this.trk_1_time = null;
    }
    // Track #2
    if (current_time - this.trk_2_time > 4 && current_time -
this.trk_2_time < 100000) {
        this.sig_4w = false;
        this.sig_4e = false;

        this.route_w_trk_2 = null;
        this.route_e_trk_2 = null;
        this.routed_trk_2 = false;

        this.trk_2_occupied = false;
        this.trk_2_time = null;
    }
}
// ---- END can_clear() ----

/**
 * throw_sw_1()
 * @summary 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() {
    if (this.sw_1 === false) {
        this.sw_1 = true;
    }
    else {
        this.sw_1 = false;
    }
}
// ---- END throw_sw_1() ----

/**
 * throw_sw_3()
 * @summary 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() {
        if (this.sw_3 === false) {
            this.sw_3 = true;
        }
        else {
            this.sw_3 = false;
        }
    }
}
// ---- END throw_sw_3() ----

/**
 * throw_sw_5()
 * @summary 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() {
    if (this.sw_5 === false) {
        this.sw_5 = true;
    }
    else {
        this.sw_5 = false;
    }
}
// ---- END throw_sw_5() ----

/**
 * get_routes()
 * @summary Gets all the routes from the interlocking
 *
 * @returns An Array holding every route variable from the
interlocking
 */
get_routes() {
    let routes = [
        this.route_w_trk_1, this.route_w_trk_2,
this.route_w_trk_3,
        this.route_e_trk_1, this.route_e_trk_2
    ];

    return routes;
}
// ---- END get_routes() ----

/**
 * get_interlocking_status()
 * @summary returns the status of the interlocking that would be

```

needed by the ReactJS Components

```

    *
    * @description 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
    *
    * @returns Object with the status of the interlocking
    */
    get_interlocking_status() {
        let status = {
            sw_1: this.sw_1,
            sw_3: this.sw_3,
            sw_5: this.sw_5,
            routes: this.get_routes(),
            routed_trk_1: this.routed_trk_1,
            routed_trk_2: this.routed_trk_2,
            occupied_trk_1: this.trk_1_occupied,
            occupied_trk_2: this.trk_2_occupied
        }

        return status;
    }
    // ---- END get_interlocking_status() ----
}

// This is required when using ReactJS
export default CTC_PA;
```