

# **Modeling UI with Statecharts**

Kelli Rockwell

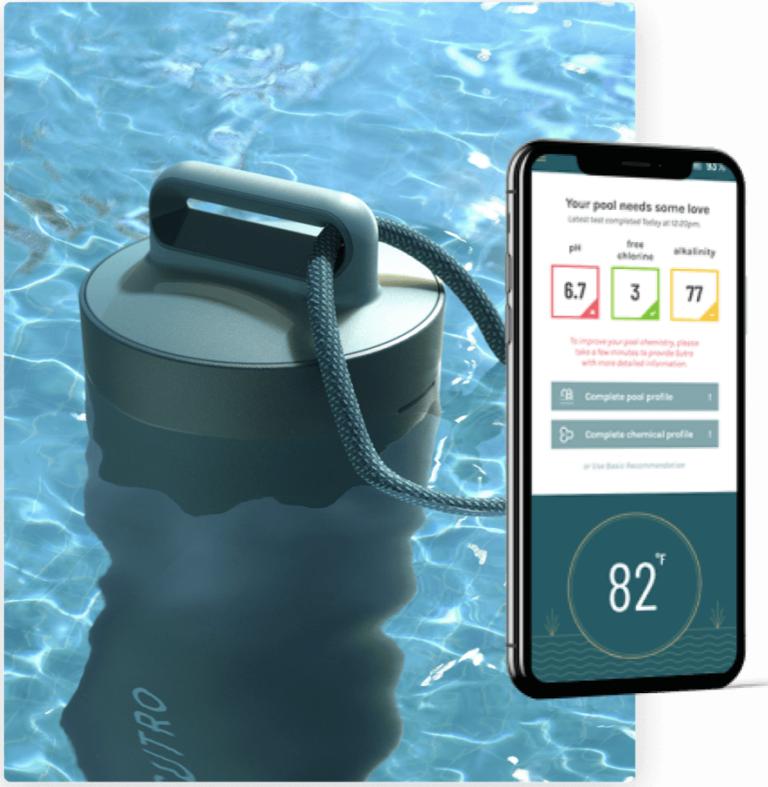
**We as developers and designers are  
really good at making the  
impossible, possible.**

**Sometimes, we're so good at it, we  
do it unintentionally!**

mysutro.com

SUTRO

Home How it Works Why Sutro About FAQ Partners Try Now



**How It Works**

The Sutro Smart System is the Simple, Safe, and Seamless way to ensure that your pool (or spa) is ready when you are. Our patented smart water monitoring technology takes the hassle out of pool or spa water maintenance.

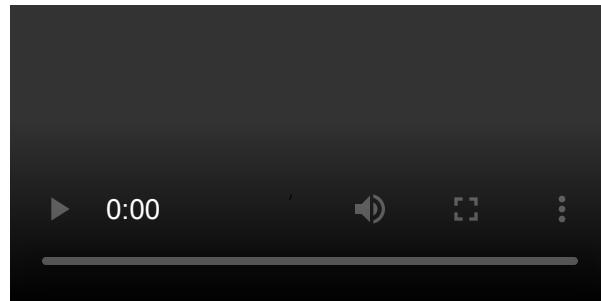
With Sutro, you can measure, monitor, and get recommendations on treatment options without having to leave the comfort of your home. With Sutro, you get what you need, when you need it, from partners that you trust.

**Sutro Smart Monitor** ▾

**Your Waters Condition** ▾

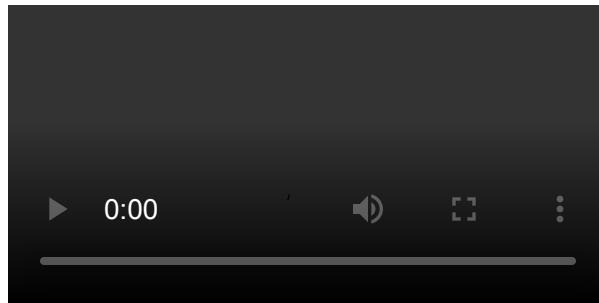
**Trusted Recommendations** ▾





\*Dramatization, not actual user recording

■ ■ ■



\*Dramatization, not actual user recording

**These are both "impossible states."  
So how did the user end up here?**

# How did the user end up here?

- We changed the application logic and the UI got out of sync

# How did the user end up here?

- We changed the application logic and the UI got out of sync
- We tried to take a shortcut

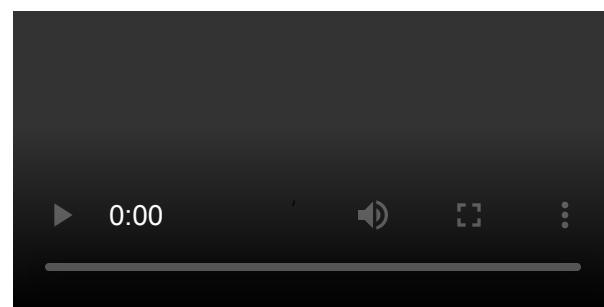
Default case is supposedly impossible to reach:

```
1 switch(calibration.step) {  
2     case "PRECHECK1":  
3         return <PreCheck1 />;  
4     case "PRECHECK2":  
5         return <PreCheck2Confirmation />;  
6     // This screen shouldn't be visible on other steps  
7     default:  
8         return <Text>{ "Calibration in progress" }</Text>;  
9     }  
10
```

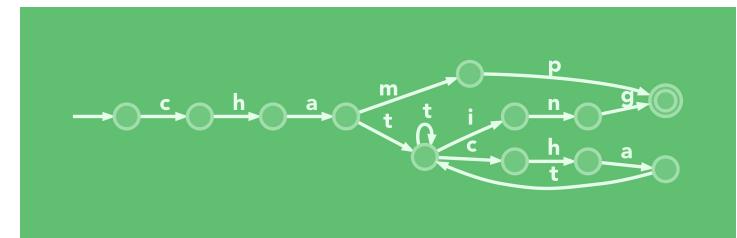
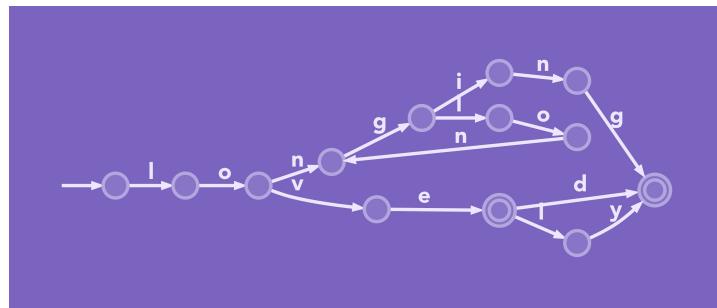
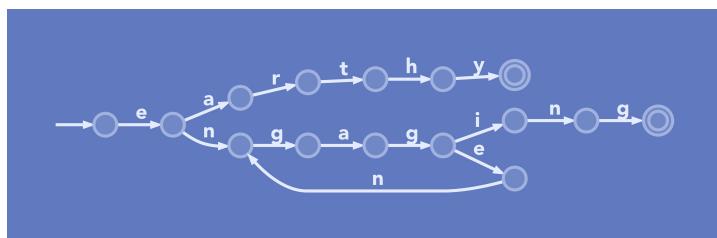
# How did the user end up here?

- We changed the application logic and the UI got out of sync
- We tried to take a shortcut
- We forgot to reset a condition
- We missed an edge case
- We didn't handle an API response properly
- ...

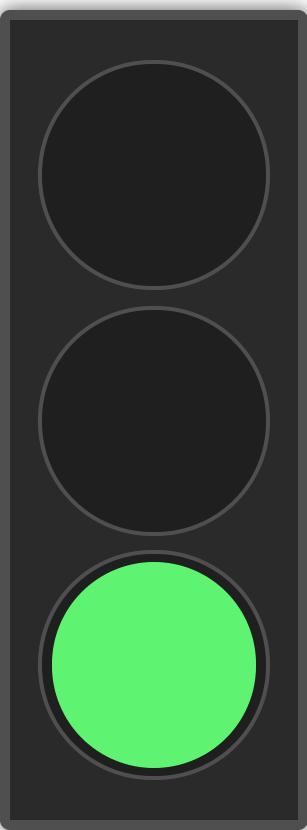
**We need a better model for our state.**



# Finite State Machines!



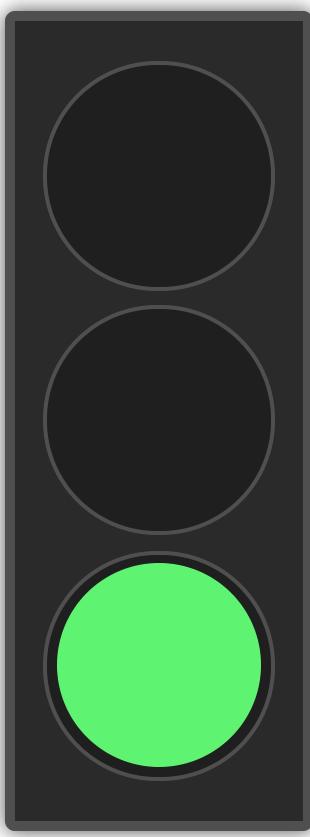
**We're already used to thinking in state machines.**



Next

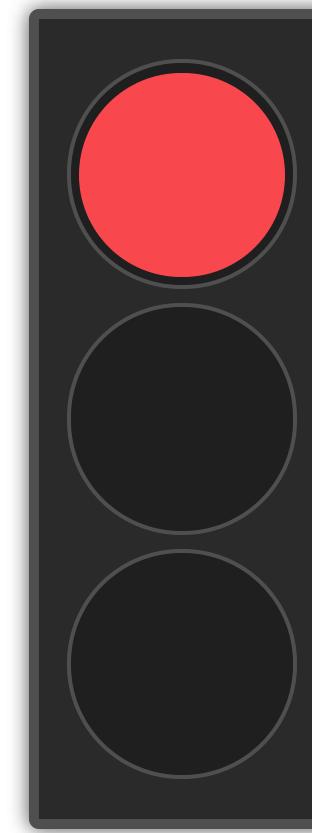
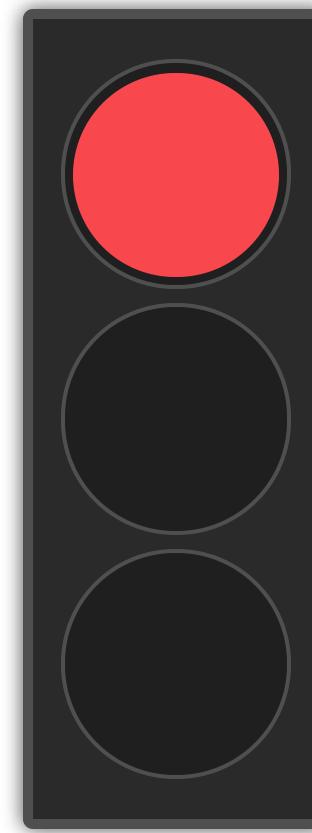
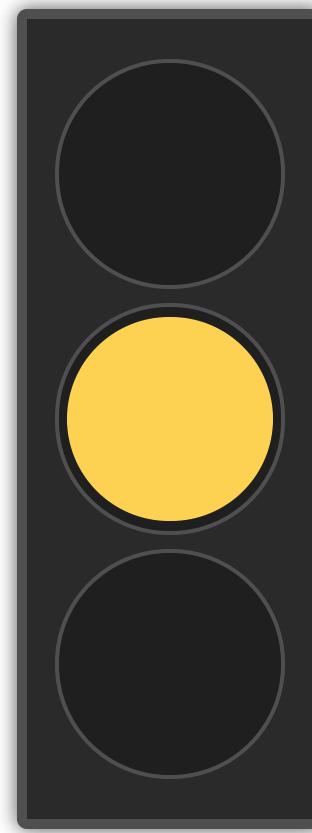
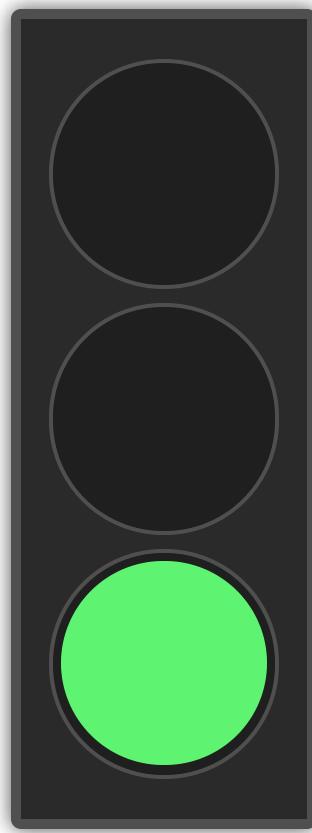
# **First rule of Finite State Machines:**

**You can only be in one state at any given time.**



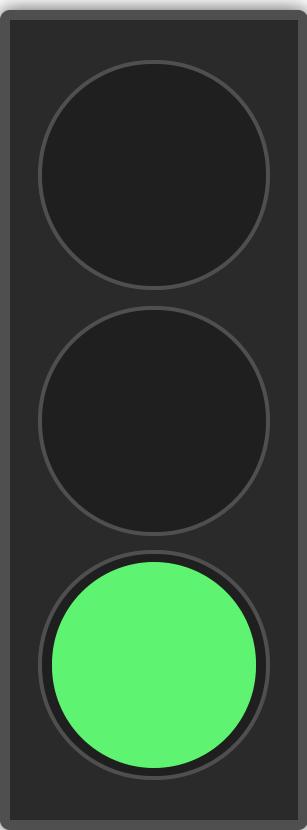
# **Second rule of Finite State Machines:**

**You have a finite number of discrete states.**



# **Third rule of Finite State Machines:**

**You transition from one state to another based on an event.**



Next

**As a driver,**

**Given, the light is red,**

**When, 30 seconds have elapsed,**

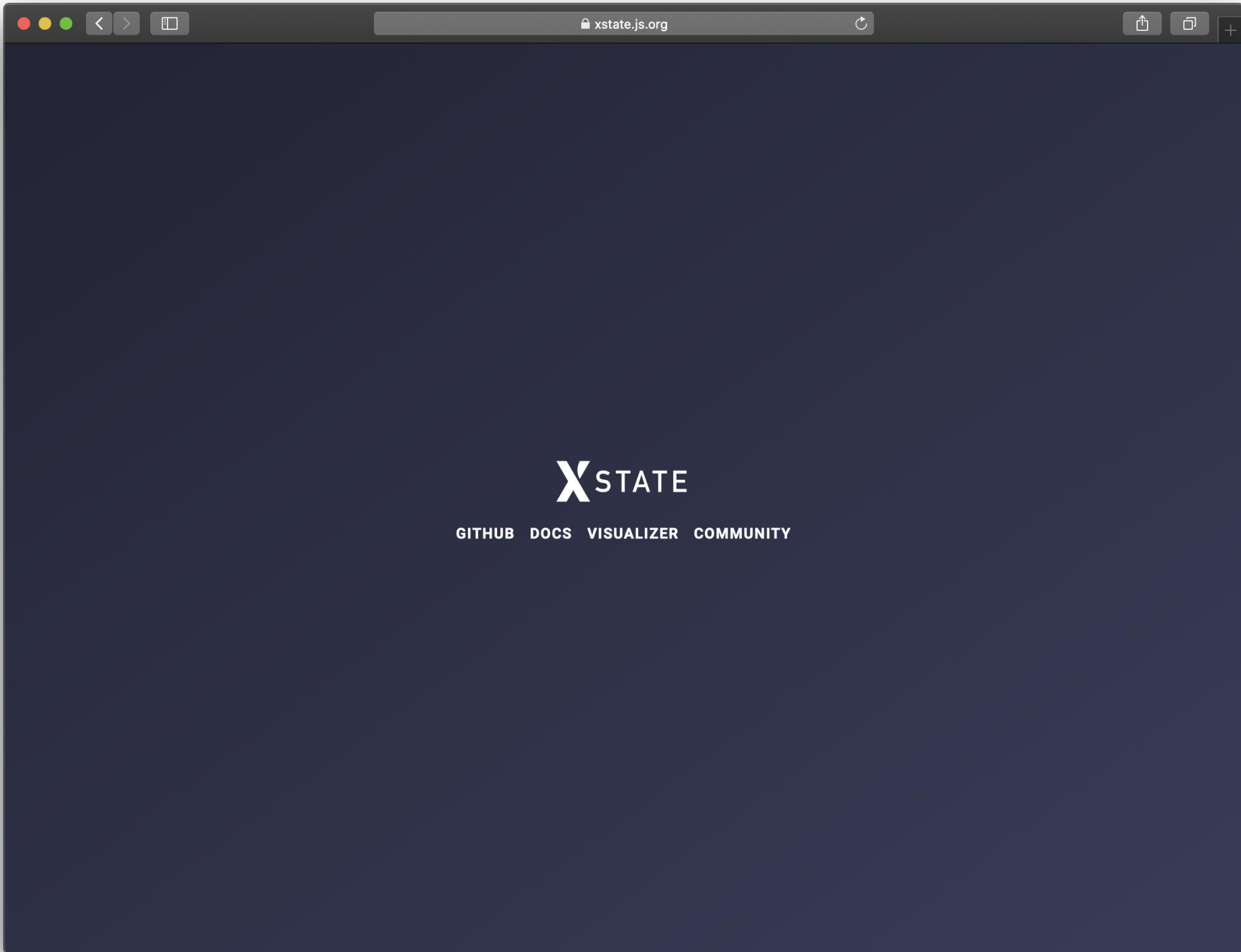
**Then, the light turns green.**



```
1 const TRANSITION_SCHEMA = {
2   green: { timer: 'yellow' },
3   yellow: { timer: 'red' },
4   red: { timer: 'green' },
5 };
6
7 class StateMachine {
8   constructor(initialState, transitions) {
9     this.state = initialState;
10    this.transitions = transitions;
11  }
12
13  transition(event) {
14    this.state = this.transitions[this.state][event];
15  }
16}
17
18 const machine = new StateMachine('green', TRANSITION_SCHEMA);
19
20 machine.transition('timer');
21 console.log(machine.state); // => 'yellow'
22
23 machine.transition('timer');
24 console.log(machine.state); // => 'red'
25
26 machine.transition('timer');
27 console.log(machine.state); // => 'green'
28
```

The problem here isn't the switch statement, it's that **the state machine is implicit**

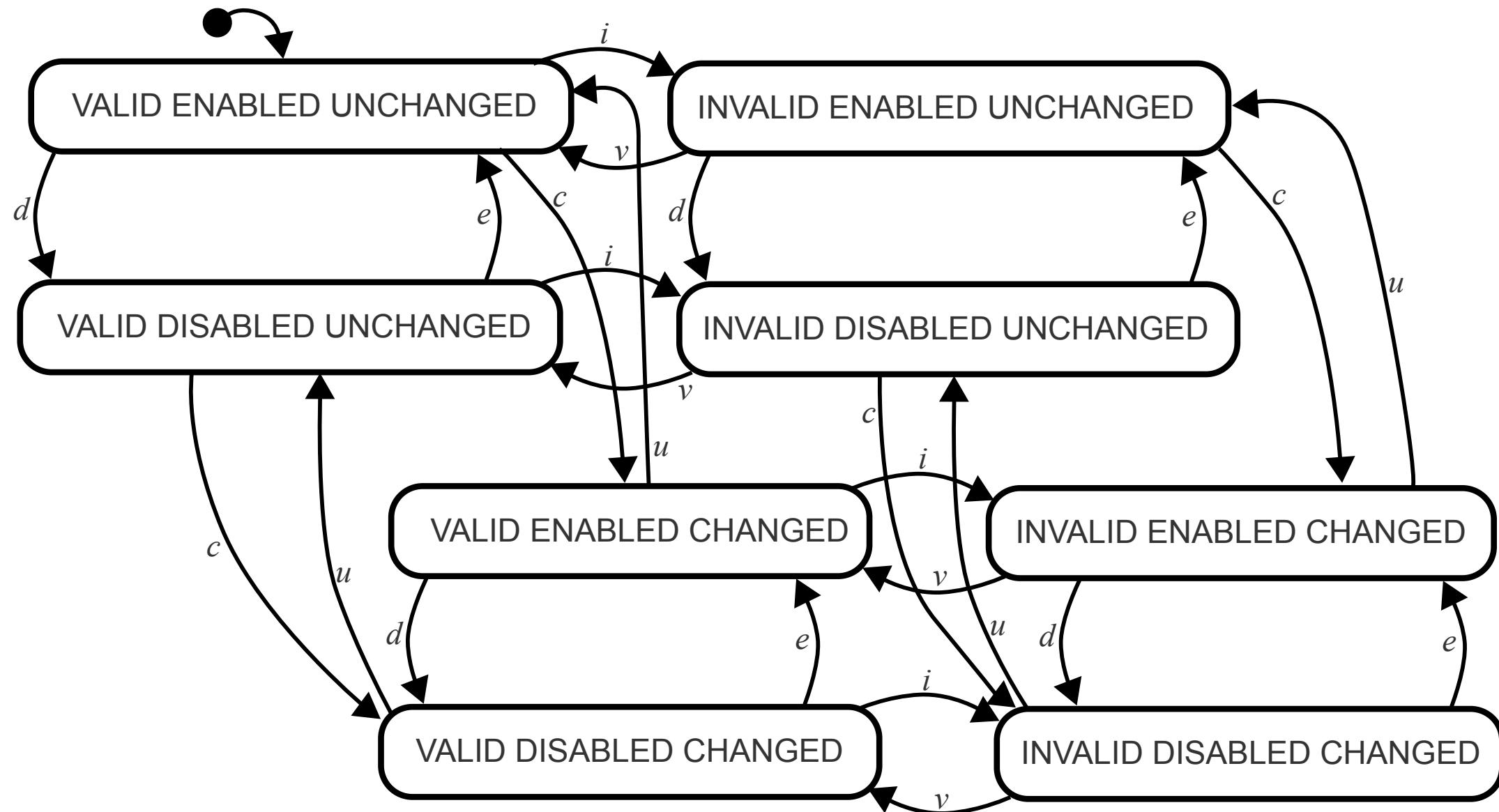
```
1 switch(calibration.step) {  
2     case "PRECHECK1":  
3         return <PreCheck1 />;  
4     case "PRECHECK2":  
5         return <PreCheck2Confirmation />;  
6     // This screen shouldn't be visible on other steps  
7     default:  
8         return <Text>{"Calibration in progress"}</Text>;  
9     }  
10 }
```

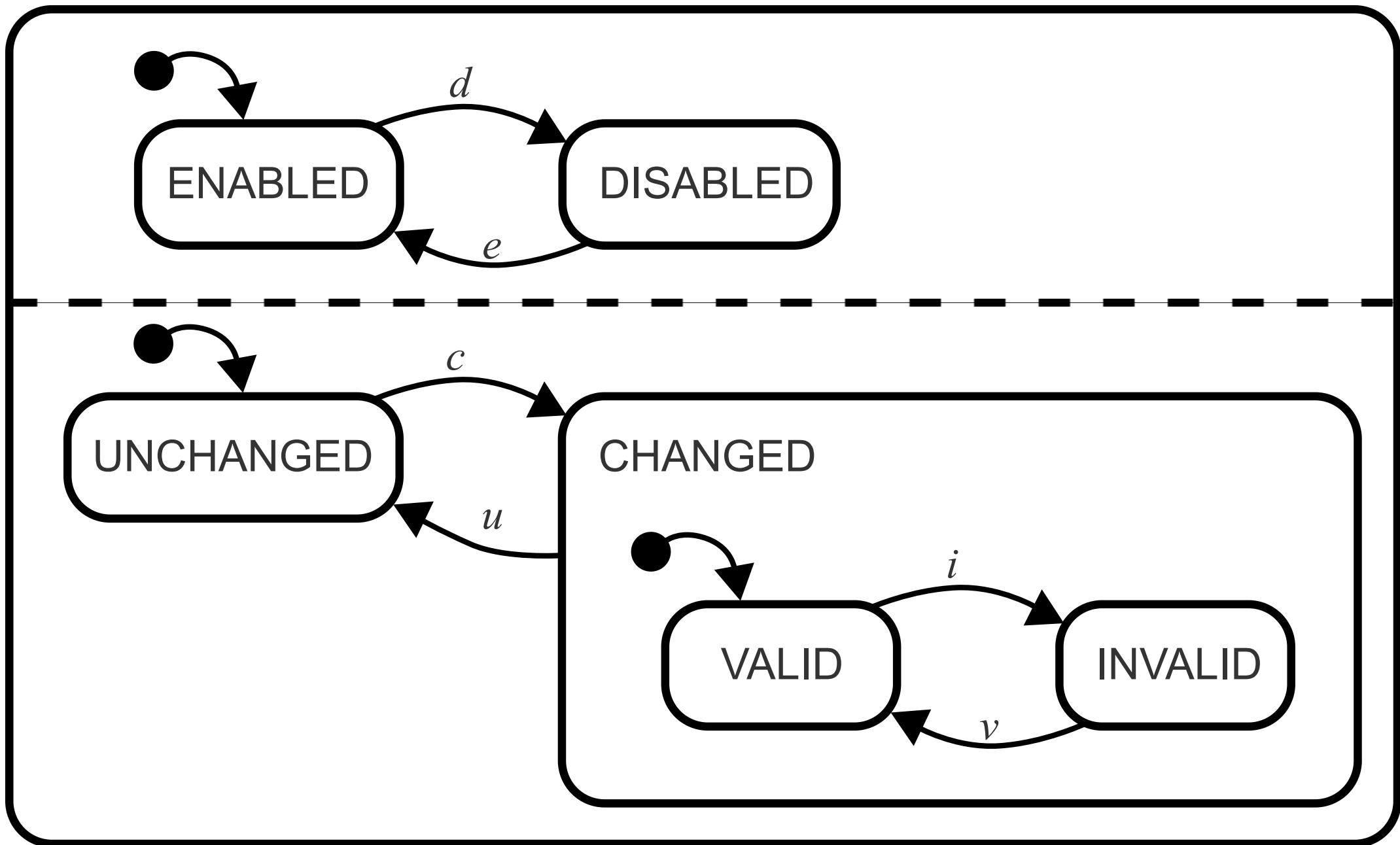


```
1 const trafficLightMachine = Machine( {
2   id: 'trafficLight',
3   initial: 'green',
4   states: {
5     green: {
6       on: { TIME_UP: 'yellow' },
7     },
8     yellow: {
9       on: { TIME_UP: 'red' },
10    },
11    red: {
12      on: { TIME_UP: 'green' },
13    },
14  },
15}) ;
16
```

```
1 const expandedTrafficLightMachine = Machine( {
2   id: 'expandedTrafficLight',
3   initial: 'connected',
4   states: {
5     disconnected: {
6       on: { SIGNAL_FOUND: 'connected' },
7     },
8     connected: {
9       initial: 'green',
10      on: { SIGNAL_LOST: 'disconnected' },
11      states: {
12        green: {
13          on: { TIME_UP: 'yellow' },
14        },
15        yellow: {
16          on: { TIME_UP: 'red' },
17        },
18        red: {
19          on: { TIME_UP: 'green' },
20        },
21      },
22    },
23  },
24} );
```

**Statechart = finite state machine  
with nested and/or parallel states**





```
1 const expandedTrafficLightMachine = Machine( {
2   id: 'expandedTrafficLight',
3   initial: 'connected',
4   states: {
5     disconnected: {
6       on: { SIGNAL_FOUND: 'connected' },
7     },
8     connected: {
9       initial: 'green',
10      on: { SIGNAL_LOST: 'disconnected' },
11      states: {
12        green: {
13          on: { TIME_UP: 'yellow' },
14        },
15        yellow: {
16          on: { TIME_UP: 'red' },
17        },
18        red: {
19          on: { TIME_UP: 'green' },
20        },
21      },
22    },
23  },
24} );
```

```
1 const TrafficLightWithControl: FC<{}> = () => {
2   const [state, send] = useMachine(expandedTrafficLightMachine);
3
4   const [currentColor, setCurrentColor] = useState<StateColor>('green');
5
6   const getColor = (state: ExpandedState): StateColor => {
7     switch (true) {
8       case state.matches({ connected: 'green' }):
9         return 'green';
10      case state.matches({ connected: 'yellow' }):
11        return 'yellow';
12      case state.matches({ connected: 'red' }):
13        return 'red';
14      case state.matches('disconnected'):
15        return 'hazard';
16    }
17  };
18
19 // When the state changes, set the color of the traffic light to match
20 useEffect(() => {
21   setCurrentColor(getColor(state));
22 }, [state]);
23
24 // Sends a "SIGNAL_FOUND" or "SIGNAL_LOST" event depending on the current state
25 const toggleSignal = () =>
26   state.matches('disconnected') ? send('SIGNAL_FOUND') : send('SIGNAL_LOST');
27
28 const buttons = (
29   <div style={styles.flex}>
30     <button onClick={() => send('TIME_UP')}>Next</button>
31     <button onClick={toggleSignal} style={styles.invisible} />
32   </div>
33 );
34
35 return (
36   <>
37     <TrafficLight currentColor={currentColor} />
38     {buttons}
39   </>
40 );
41 };
42 }
```

**Enough with the traffic lights  
already!**

### MACHINE-0.TS

```
1 export const calibrationMachine = Machine({
2   id: 'machine',
3   initial: 'dashboard',
4   states: {
5     dashboard: {
6       on: { START_CALIBRATION: 'precheck1' },
7     },
8     precheck1: {
9       on: { PASS_PRECHECK1: 'precheck2' },
10    },
11    precheck2: {
12      on: { PASS_PRECHECK2: 'prime' },
13    },
14    prime: {
15      on: { PASS_PRIME: 'dashboard' },
16    },
17  },
18 });
19
```

### APP-0.TSX

```
1 const getView = (
2   state: StateValue,
3   send: (event: CalibrationEvent) => void,
4 ): JSX.Element[] => {
5   switch (state) {
6     case 'dashboard': {
7       return [
8         <h2>Dashboard</h2>,
9         <button onClick={() => send({ type: 'START_CALI' >
10           Start Calibration
11         </button>,
12       ];
13     }
14     case 'precheck1': {
15       return [
16         <h2>Precheck1</h2>,
17         <button onClick={() => send({ type: 'PASS_PREC' >
18           Pass Precheck1
19         </button>,
20       ];
21     }
22     case 'precheck2': {
23       return [
24         <h2>Precheck2</h2>,
25         <button onClick={() => send({ type: 'PASS_PRIM' >
26           Pass Prime
27         </button>,
28       ];
29     }
30   }
31 }
```

# Dashboard

Start Calibration

0

### MACHINE-1.TS

```
1  /** Builds a polling activity for a given polling func
2 const generatePollingActivity = (pollingFn: () => void)
3   // Start the polling activity
4   const interval = setInterval(pollingFn, 1000);
5
6   // Return a function that stops the polling activity
7   return () => {
8     console.log('stopping poll');
9     clearInterval(interval);
10  };
11 }
12
13 export const calibrationMachine = Machine(
14   {
15     id: 'machine',
16     initial: 'dashboard',
17     states: {
18       dashboard: {
19         on: {
20           START_CALIBRATION: 'precheck1',
21         },
22       },
23     },
24   }
25 );
```

### APP-1.TSX

```
1 const getView = (
2   state: CalibrationState,
3   send: (event: CalibrationEvent) => void,
4 ): JSX.Element[] => {
5   switch (true) {
6     case state.matches('dashboard'): {
7       return [
8         <h2>Dashboard</h2>,
9         <button onClick={() => send({ type: 'START_CALIBRATION' })}>
10           Start Calibration
11         </button>,
12       ];
13     }
14     case state.matches('precheck1'): {
15       return [
16         <h2>Precheck1</h2>,
17         <button onClick={() => send({ type: 'PASS_PRECHECK1' })}>
18           Pass Precheck1
19         </button>,
20         getRetryButton(state, send),
21       ];
22     }
23   }
24 }
```

# Dashboard

Start Calibration

Error

### MACHINE-2.TS

```
1 export type CalibrationContextStatus = 'COMPLETE' | 'IN_PROGRESS' | 'PRECHECK1' | 'PRECHECK2';
2 export type CalibrationContextStep = 'PRECHECK1' | 'PRECHECK2' | 'CALIBRATION' | 'DONE';
3
4
5 /** The context (extended state) of the machine */
6 export type CalibrationContext = {
7   needsCalibration: boolean;
8   status: CalibrationContextStatus;
9   step: CalibrationContextStep;
10};
11
12
13 /** Builds a polling activity for a given polling function */
14 const generatePollingActivity = (pollingFn: () => void) => {
15   // Start the polling activity
16   const interval = setInterval(pollingFn, 1000);
17
18   // Return a function that stops the polling activity
19   return () => {
20     console.log('stopping poll');
21     clearInterval(interval);
22   };
23};
```

### APP-2.TSX

```
1 const getView = (
2   state: CalibrationState,
3   send: (event: CalibrationEvent) => void,
4 ): JSX.Element[] => {
5   switch (true) {
6     case state.matches('dashboard'):
7       const disabled = !state.context.needsCalibration;
8       const button =
9         state.context.status === 'IN_PROGRESS' ? (
10           <button
11             disabled={disabled}
12             key="resume"
13             onClick={() => send({ type: 'RESUME' })}
14           >
15             Resume
16           </button>
17         ) : (
18           <button
19             disabled={disabled}
20             key="button"
21             onClick={() => send({ type: 'START_CALIBRATION' })}
22           >
23             Start Calibration
24           </button>
25         );
26       return [button];
27     default:
28       return [];
29   }
30};
```

# Dashboard

Start Calibration

Error

## Initial Context from the "Server"

Status: complete

Step: done

Needs Calibration?

```
{
  "needsCalibration": false,
  "status": "COMPLETE",
  "step": "DONE"
}
```

### MACHINE-3.TS

```
1 export type CalibrationContextStatus = 'COMPLETE' | 'IN_PROGRESS' | 'FAILED';
2 export type CalibrationContextStep = 'PRECHECK1' | 'PRECHECK2' | 'CALIBRATION';
3
4
5 /** The context (extended state) of Precheck1 */
6 export type Precheck1Context = {
7   firmwareUpToDate: 'waiting' | boolean;
8   lidClosed: 'waiting' | boolean;
9   batteryCharged: 'waiting' | boolean;
10  cartridgeInstalled: 'waiting' | boolean;
11 };
12
13 export const INITIAL_PRECHECK1_CONTEXT: Precheck1Context = {
14   firmwareUpToDate: 'waiting',
15   lidClosed: 'waiting',
16   batteryCharged: 'waiting',
17   cartridgeInstalled: 'waiting',
18 };
19
20
21 /** The context (extended state) of the machine */
22 export type CalibrationContext = {
23   needsCalibration: boolean;
24 }
```

### APP-3.TSX

```
1 const getDisplayText = (contextValue: 'waiting' | boolean) => {
2   switch (contextValue) {
3     case 'waiting':
4       return '?';
5     case true:
6       return 'PASS';
7     case false:
8       return 'FAIL';
9   }
10 };
11
12 const Precheck1: FC<Precheck1Context & { send: (event: string, value: string) => void }> = ({ send, ...rest }) => {
13   const [firmwareUpToDate, setFirmwareUpToDate] = useState('waiting');
14   const [lidClosed, setLidClosed] = useState('waiting');
15   const [batteryCharged, setBatteryCharged] = useState('waiting');
16   const [cartridgeInstalled, setCartridgeInstalled] = useState('waiting');
17
18   const handleSend = (event: string, value: string) => {
19     if (value === 'true') {
20       setFirmwareUpToDate('true');
21     } else if (value === 'false') {
22       setFirmwareUpToDate('false');
23     }
24   };
25
26   const handleLidClosed = (value: string) => {
27     if (value === 'true') {
28       setLidClosed('true');
29     } else if (value === 'false') {
30       setLidClosed('false');
31     }
32   };
33
34   const handleBatteryCharged = (value: string) => {
35     if (value === 'true') {
36       setBatteryCharged('true');
37     } else if (value === 'false') {
38       setBatteryCharged('false');
39     }
40   };
41
42   const handleCartridgeInstalled = (value: string) => {
43     if (value === 'true') {
44       setCartridgeInstalled('true');
45     } else if (value === 'false') {
46       setCartridgeInstalled('false');
47     }
48   };
49
50   const handleSendCalibration = () => {
51     send('CALIBRATION', 'true');
52   };
53
54   const handleSendPrecheck1 = () => {
55     send('PRECHECK1', 'true');
56   };
57
58   const handleSendPrecheck2 = () => {
59     send('PRECHECK2', 'true');
60   };
61
62   const handleSendComplete = () => {
63     send('COMPLETE', 'true');
64   };
65
66   const handleSendFailed = () => {
67     send('FAILED', 'true');
68   };
69
70   const handleSendInProgress = () => {
71     send('IN_PROGRESS', 'true');
72   };
73
74   const handleSendUnknown = () => {
75     send('UNKNOWN', 'true');
76   };
77
78   const handleSendWaiting = () => {
79     send('WAITING', 'true');
80   };
81
82   const handleSendTrue = () => {
83     send('TRUE', 'true');
84   };
85
86   const handleSendFalse = () => {
87     send('FALSE', 'false');
88   };
89
90   const handleSendUnknownBoolean = () => {
91     send('UNKNOWN_BOOLEAN', 'true');
92   };
93
94   const handleSendUnknownString = () => {
95     send('UNKNOWN_STRING', 'true');
96   };
97
98   const handleSendUnknownNumber = () => {
99     send('UNKNOWN_NUMBER', 'true');
100 };
101
102   return (
103     <div>
104       <h1>Machine Status</h1>
105       <table>
106         <thead>
107           <tr>
108             <th>Parameter</th>
109             <th>Value</th>
110           </tr>
111         </thead>
112         <tbody>
113           <tr>
114             <td>Firmware Up-to-date</td>
115             <td>{firmwareUpToDate}</td>
116           </tr>
117           <tr>
118             <td>Lid Closed</td>
119             <td>{lidClosed}</td>
120           </tr>
121           <tr>
122             <td>Battery Charged</td>
123             <td>{batteryCharged}</td>
124           </tr>
125           <tr>
126             <td>Cartridge Installed</td>
127             <td>{cartridgeInstalled}</td>
128           </tr>
129         </tbody>
130       </table>
131       <button onClick={handleSendCalibration}>Calibrate</button>
132       <button onClick={handleSendPrecheck1}>Precheck 1</button>
133       <button onClick={handleSendPrecheck2}>Precheck 2</button>
134       <button onClick={handleSendComplete}>Complete</button>
135       <button onClick={handleSendFailed}>Failed</button>
136       <button onClick={handleSendInProgress}>In Progress</button>
137       <button onClick={handleSendUnknown}>Unknown</button>
138       <button onClick={handleSendWaiting}>Waiting</button>
139       <button onClick={handleSendTrue}>True</button>
140       <button onClick={handleSendFalse}>False</button>
141       <button onClick={handleSendUnknownBoolean}>Unknown Boolean</button>
142       <button onClick={handleSendUnknownString}>Unknown String</button>
143       <button onClick={handleSendUnknownNumber}>Unknown Number</button>
144     </div>
145   );
146 };
147
148 export default Precheck1;
```

# Dashboard

Start Calibration

Fail Chance

50

# **Takeaways**

# Takeaways

- Keeping UI in sync with application logic is universally hard

# Takeaways

- Keeping UI in sync with application logic is universally hard
- Implicit state machines make it harder

# Takeaways

- Keeping UI in sync with application logic is universally hard
- Implicit state machines make it harder
- You're probably already thinking in state machines

# Takeaways

- Keeping UI in sync with application logic is universally hard
- Implicit state machines make it harder
- You're probably already thinking in state machines
- Statecharts = beefed up state machines

# Takeaways

- Keeping UI in sync with application logic is universally hard
- Implicit state machines make it harder
- You're probably already thinking in state machines
- Statecharts = beefed up state machines
- **Modeling UI with statecharts is something you should consider doing  
:)**

**Thank you!**