Applications Note: Automated Two-Train Operation Using Passing Sidings

Overview:

This Applications Note describes the automated operation of two trains running on a single shared mainline using a passing siding.

The first train departs from the station, makes one complete circuit around the mainline, returns to the station, and stops. The second train then departs in the opposite direction, makes one complete circuit around the mainline, returns to the station, and stops. The action then repeats.

Track Layout:

The track layout for this example is shown in Figure 1. Common ground wiring is employed. Three insulated track blocks, here labeled A, B, and C are used.

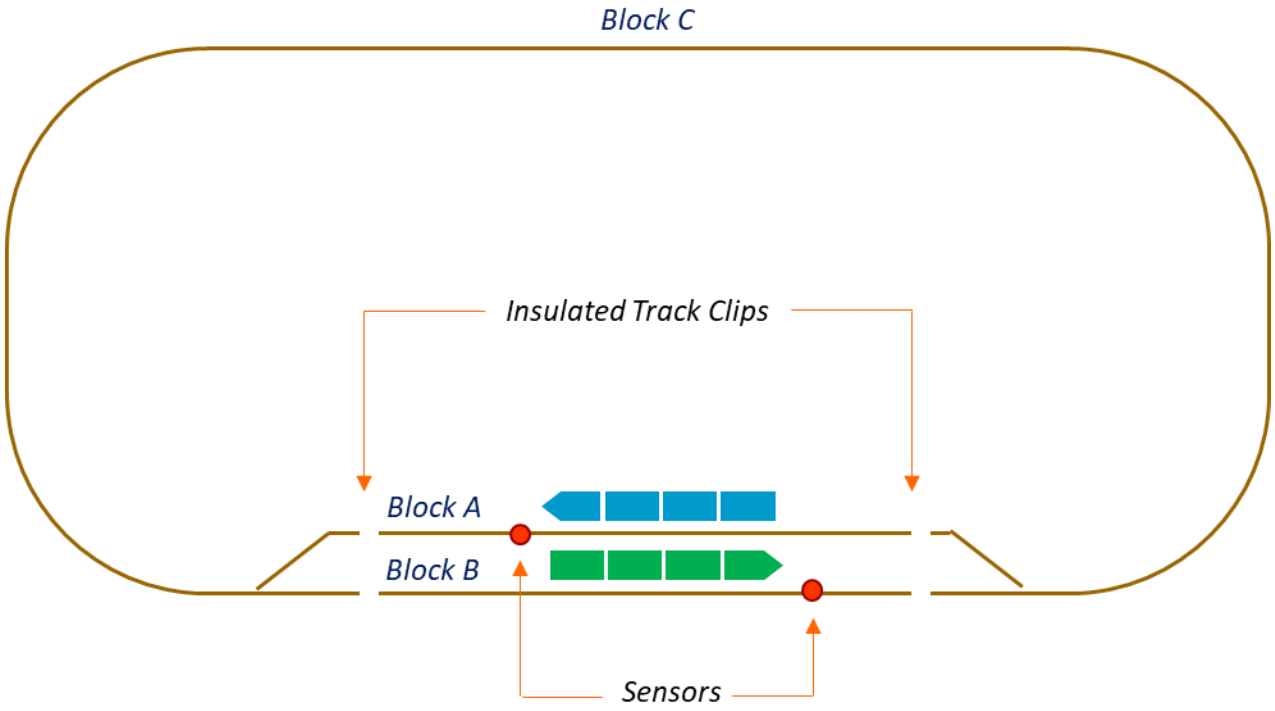

Figure 1. Track Layout and Sensor Positioning

CTI Hardware:

The automated operation of the two trains requires one Train Brain and one Smart Cab module.

Three of the Train Brain's four controllers and two of its four sensor ports are utilized. The two modules are wired as shown in Figure 2.

In this example, we assume the use of magnetic sensors, however, the example can be easily adapted to work with all sensor types. The sensors are situated as shown in the track diagram of Figure 1.

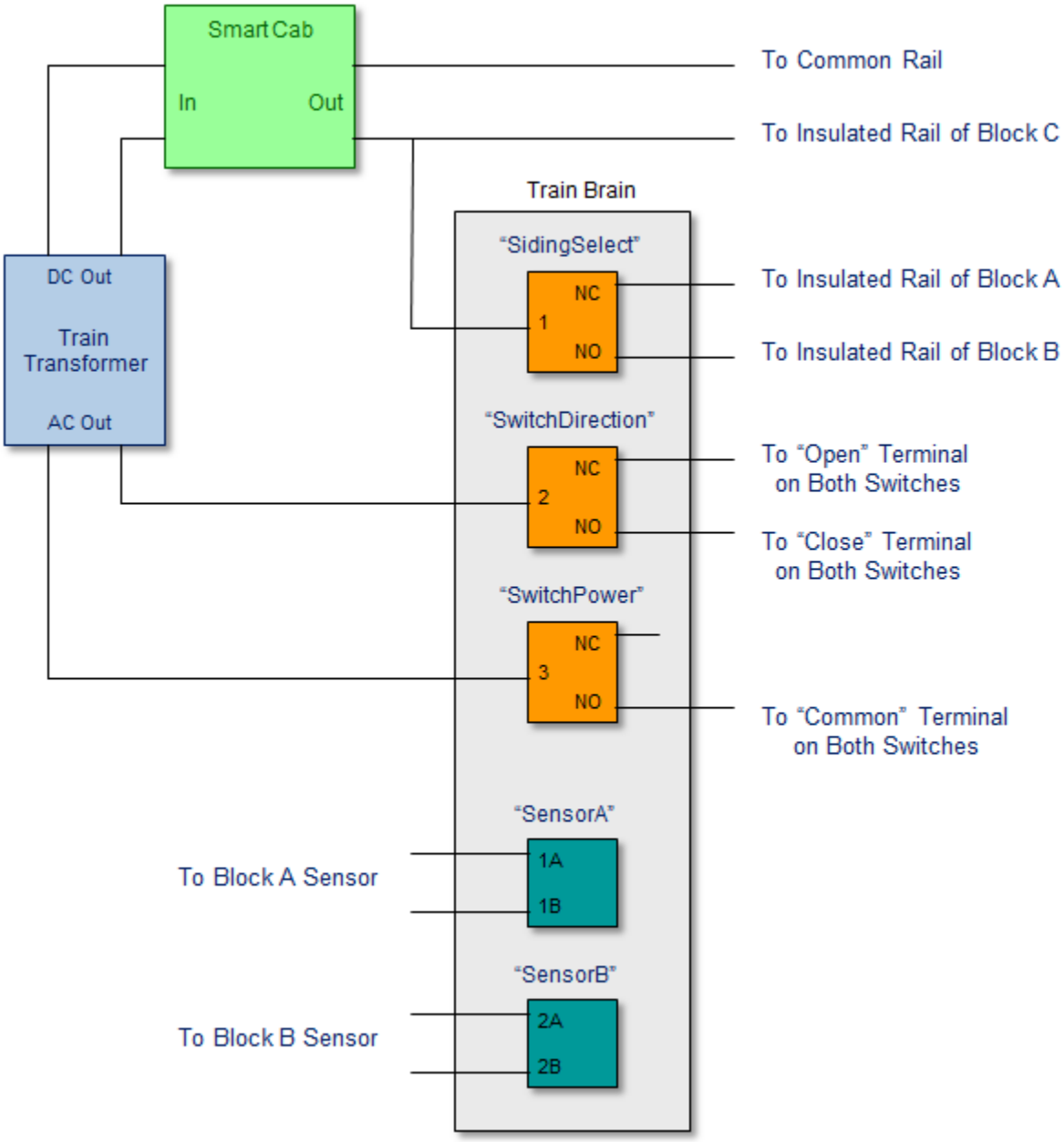

Figure 2. CTI Module Wiring Diagram

TCL Programming:

As always, it's best to begin with an "English language" description of what we want our TCL program to accomplish. Then, we'll translate that description into the more formal TCL language syntax that the TBrain program understands. With that in mind, here's a description of what we want our TCL program to do:

> *Our TCL program should monitor each of our two sensors to detect the arrival of a train at the station. Each time a train arrives, we want to bring it to a smooth stop by applying the brake on our throttle. We then want to remove power from the siding on which the train arrived, and route it instead to the siding occupied by the other train. We'll then throw the switches to route the newly selected siding to the mainline. Finally, we'll release the brake on our throttle, allowing the newly selected train to depart.*

To assist us in writing our TCL program, it's often helpful to first reorganize our written description into a more structured list form, as follows:

When a train is detected arriving at the station, do the following: (a)
1) *Apply the brake to bring that train to a smooth stop* (b)
2) *Route power to the opposite passing siding* (c)
3) *Throw the switches to route the newly selected siding to the mainline* (d)
4) *Release the brake to allow the train on the newly selected siding to depart* (e)

With our thoughts organized in list form, it's a rather straightforward task to write our TCL program. In fact, the entire automated operation of the two trains requires only a single TCL language *When-Do* statement.

The resulting TCL program is shown below. As you can see, the TCL code looks very much like our English language list. [To illustrate that point, we've given each line in the list above an index, (a) through (e). Then, we've shown that index next to the statement(s) in our TCL program corresponding to that item. These indices are not part of the actual TCL program.]

```
Controls:  SidingSelect, SwitchDirection, SwitchPower, spare
Sensors:   SensorA, SensorB, spare, spare
SmartCabs: Cab1

Actions:

      When SensorA = True Or SensorB = True Do          (a)
          Cab1.Brake = On,                              (b)
          Wait 10,                                      (b)
          SidingSelect = SidingSelect~                  (c)
          SwitchDirection = SwitchDirection~            (c)
          SwitchPower = Pulse 0.25                      (d)
          Cab1.Direction = Cab1.Direction~              (e)
          Cab1.Brake = Off                              (e)
```

That's all it takes to automate the operation of our trains. Here's how it works:

Our TCL program's single *When-Do* statement is activated whenever either of our sensors is triggered, using the *When* clause:

```
When SensorA = True Or SensorB = True Do
```

Based on the positions of our sensors, this condition occurs any time a train arrives at the station. (The locations of the two magnetic sensors may need to be adjusted based on the length of the train and siding to allow enough room for the train to fully stop within the siding for the chosen momentum setting.)

In response, we immediately carry out our When-Do statement's first action, stopping the arriving train by applying the brake on our Smart Cab, using the statement:

```
Cab1.Brake = On
```

We then wait 10 seconds to allow the Smart Cab's simulated momentum feature to bring the train to a smooth, prototypical stop.

Once the train has come to a complete stop, we route power away from its siding and onto the siding occupied by the other train. We accomplish that task using the statement:

```
SidingSelect = SidingSelect~
```

Recall from our wiring diagram that the Train-Brain routes power to either siding A or siding B, depending on the state of its "*SidingSelect*" control relay. When *SidingSelect* is "Off" (i.e. when its relay is in the NC position) the Smart Cab's output is routed to block A. When *SidingSelect* is "On" (i.e. when its relay is in the NO position), the Smart Cab's output is routed to block B.

Thus, each time a train arrives, we want to change the state of *SidingSelect* to be the opposite of its current state. When it's On, we want to turn it Off. When it's Off, we want to turn it On.

We can accomplish that very easily, using TCL's logical "not" operator '~', which produces the opposite logical state of its operand. Thus, the above action statement sets *SidingSelect* equal to the opposite of its current logical state, as desired.

Now it's time to throw our turnouts to route the newly selected siding to the mainline. For dual-coil solenoid driven switch machines, that's a two-step process. (See Section 5 of the *CTI User's Guide* for a full discussion of switch control if you want to try this example using other types of switch machines.)

First, we configure the controller governing switch direction. As before, TCL's '~' operator is used to set *SwitchDirection* to the opposite of its current logical state. Then we apply a 0.25 second power pulse via the *SwitchPower* controller to throw the turnouts. (Note that since we always want our two switches to be thrown in tandem, they're both controlled using the same Train-Brain controllers.)

```
SwitchDirection = SwitchDirection~
SwitchPower = Pulse 0.25
```

Finally, we're now ready to send our second train on its way. Since it will be traveling in the opposite direction of our first train, we'll first need to change the direction on our Smart Cab. (You guessed it, we'll just use TCL's "~" operator to accomplish that, too.) Then we simply release the brake, and off she goes. Here's the TCL code to do it:

```
Cab1.Direction = Cab1.Direction~
Cab1.Brake = Off
```

As our second train begins its journey, the operation of our When-Do statement is complete.

Once the second train finishes its circuit around the mainline, it will re-enter its passing siding at the station. And when it does, our When-Do statement will detect its arrival, and the entire process will begin again.

Enhancements:

The above TCL program works great, but one obvious drawback is that both trains run at the same throttle setting. Unless our two trains happen to be closely matched in weight and engine performance, we'd really prefer to be able to control the speed of each train independently from the other.

That's an easy enhancement to add to our program. All we'll need is a variable to remember the speed of the idled train. When a train pulls into the station, we'll save its speed setting away into this variable. Then the next time that train is ready for departure, we'll restore its speed setting to the value we previously stored away.

In this way, we can use the on-screen throttle to set the speed of each train the first time it departs (or to adjust the train's speed anytime thereafter). From then on, our automated "cruise-control" will take effect to bring each train back up to that speed again each time it leaves the station.

Here's our revised TCL program:

```
Controls:  SidingSelect, SwitchDirection, SwitchPower, spare
Sensors:   SensorA, SensorB, spare, spare
SmartCabs: Cab1
Variables: RememberedSpeed, Temp

Actions:

     When SensorA = True Or SensorB = True Do
         Cab1.Brake = On,
         Wait 10,
         SidingSelect = SidingSelect~
         SwitchDirection = SwitchDirection~
         SwitchPower = Pulse 0.25
         Temp = Cab1.Speed
         Cab1.Speed = RememberedSpeed
         RememberedSpeed = Temp
         Cab1.Direction = Cab1.Direction~
         Cab1.Brake = Off
```

Modifications:

This example can be used “as is”, or may be easily adapted to support 3 (or more) passing sidings, interactive point-and-click siding selection/train activation from an on-screen CTC panel, etc. Just use the code provided here as a starting point, and let your imagination take over from there.