Applications Note: Automated Four-Train Operations Using Cab Control

This Applications Note describes the automated operation of four trains running on a single shared mainline using computer-automated *"cab control"*.

Note: The following discussion builds upon topics introduced in our two-train cab control applications note. We recommend the reader first read and understand that example before proceeding on to the four-train case.

Operation:

Up to four trains run simultaneously along a shared mainline loop, which is divided into a number of electrically isolated track blocks. A separate, dedicated throttle is assigned to each train traveling on the mainline. Each throttle is automatically routed to follow its train as it moves from block to block, providing seamless, independent speed control of each engine.

The traffic conditions ahead of each train are continually monitored. A train is automatically brought to a smooth stop whenever it approaches too close to a train ahead, and automatically returns smoothly to its previous running speed once the track ahead has cleared.

Track Layout:

The track layout for this example is shown in Figure 1. Common ground wiring is employed. Eight insulated track blocks, here labeled A through H are used.

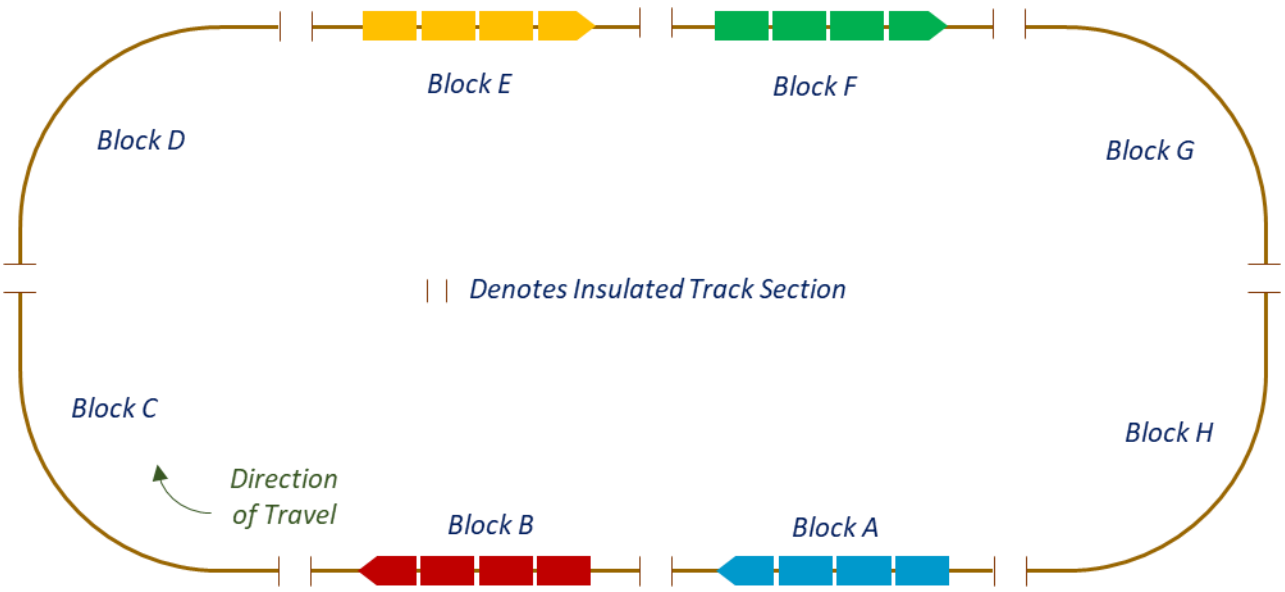

Figure 1. Track Layout

CTI Hardware:

The automated operation of the four trains requires three Dash-8's, one Watchman, and four Smart Cab modules (conventional manual throttles may also be used in lieu of the Smart Cabs).

The Dash-8 controllers are used to select the output of one of our four throttles to be routed to each track block. The Watchman's sensor ports are used to detect the presence of a train in each track block.

In this example, we assume the use of current detection sensors, however, the example may be easily adapted to work with all sensor types.

The modules are wired as shown in Figure 2.

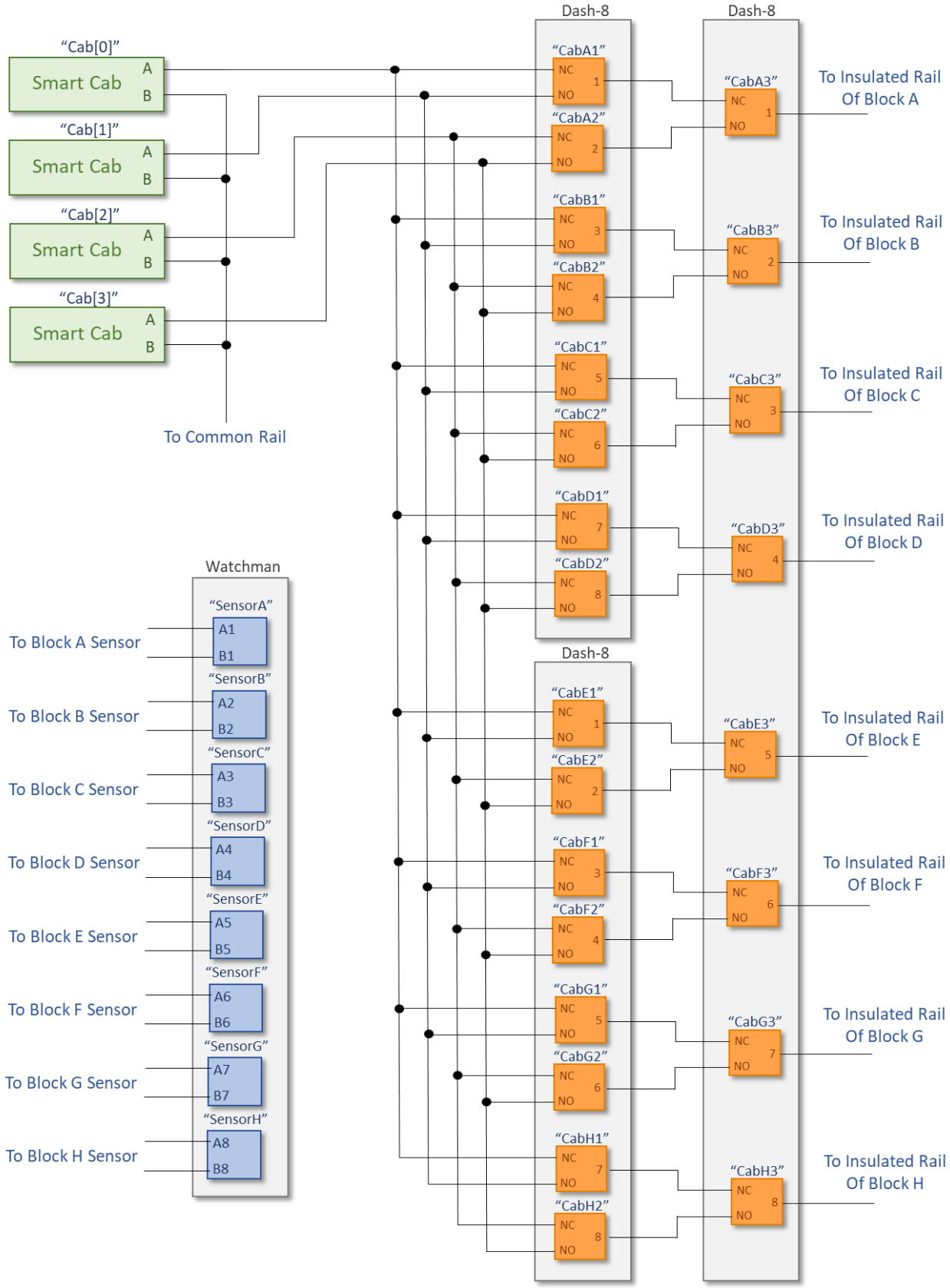

Figure 2. CTI Module Wiring Diagram

In the App Note describing a cab control scheme for operating two trains on a four block mainline, we fully described the algorithm for performing cab routing and collision avoidance, and its realization via the TCL language.

It is best to first read and fully understand the workings of that simpler case. After doing so, in this App Note, we can then concentrate our efforts on introducing the minor differences needed to extend that basic approach to a more generalized solution for operating any number of trains using cab control. In the discussion below, it is assumed that the reader has read, and understands, the earlier App Note.

As before, we will focus our discussion on a single representative track block, again block B. The behavior of all remaining track blocks will be identical, only the entity names will change.

To review, here is the English language description of our cab control "algorithm" to be performed in each track block:

1) When a train enters this block …

2) If there is traffic in the block ahead, then …

a) Apply the brake on the cab assigned to this block.

b) Wait until any traffic in the block ahead clears, then …

c) Release the brake on the cab assigned to this block.

3) Assign this block's cab to the block ahead.

And, here, for reference, is the TCL code for our earlier two-train, four-block solution (see the discussion in the earlier App Note to fully understand this TCL code):

```
1)    When SensorB = True Do
2)      If CabB = CabA Then Wait Until SensorA = False Then EndIf
3)      If SensorC = True Then
4)        Cab[CabB].Brake = On
5)        Wait Until SensorC = False Then
6)        Cab[CabB].Brake = Off
7)      EndIf
8)      CabC = CabB
```

TCL Programming Changes:

Recall that in our two-train cab control example we examined the state of our cab assignment relay controller to determine which of our two cabs was currently routed to this block (see lines 4 and 6 above). Now, however, things aren't quite that simple, since it takes three controllers to select one of our four cabs (see the wiring diagram in Figure 2).

To make the job easier, we'll create a "cab assignment" *variable*, for each track block, and use it to indicate which cab is currently assigned to that block. A value of zero in a block's cab assignment variable will indicate that Cab[0] is currently assigned to that block, a value of 1 will

indicate that Cab[1] is currently assigned to that block, etc. That way, the cab assignment variable can be used as an index into our array of four Smart Cabs, just as we did using the discrete cab control relay in our earlier example.

The four-train cab control When-Do will therefore look exactly the same! Only now, anytime we make a change to our cab assignment variables (such as in line 8 above), we'll also need to use the value of that variable to set the cab control relays to pick which of our four cabs to physically route into each track block. (Since we'll need to do that for all eight track blocks, it's the perfect opportunity to use a subroutine so we only need to do the work once.) Here's our revised cab control When-Do. The only change is highlighted in green, namely the call to our new subroutine.

```
1)    When SensorB = True Do
2)      If CabB = CabA Then Wait Until SensorA = False Then EndIf
3)      If SensorC = True Then
4)        Cab[CabB].Brake = On
5)        Wait Until SensorC = False Then
6)        Cab[CabB].Brake = Off
7)      EndIf
8)      CabC = CabB, SetRelays(&CabC1, &CabC2, &CabC3)
```

Now, here's our new "*SetRelays*" subroutine. Its purpose is to convert the cab assignment variable's value to the appropriate settings of the corresponding three cab select controllers. A simple If-Then-Else statement makes the appropriate relay settings for each of the four possible values of our *CabSelect* variable. Then, in our main program we simply pass the value of the *CabSelect* variable, and *pointers* to the relay controllers (using TCL's '&' operator) for the current track block. Remember that we need to pass *pointers* to allow the subroutine to change the state of the relays (using TCL's '*' operator) passed to it "*by reference"*. See the discussion on passing *by value* vs. *by reference* in the CTI User's Guide for more details.

```
Sub SetRelays (CabSelect, Relay1, Relay2, Relay3)
    If     CabSelect=0 Then *Relay1=Off, *Relay2=Off, *Relay3=Off
    ElseIf CabSelect=1 Then *Relay1=On,  *Relay2=Off, *Relay3=Off
    ElseIf CabSelect=2 Then *Relay1=Off, *Relay2=Off, *Relay3=On
    ElseIf CabSelect=3 Then *Relay1=Off, *Relay2=On,  *Relay3=On
    EndIf
EndSub
```

Initialization:

As in the two-train case, at startup, we can let our TCL code go out and find the starting locations of each of our trains and set the initial cab assignments.

This is a bit fancier than in the two-train case, but not by much:

```
When $Reset = True Do

  'Initialize cab assignments: engine in highest lettered track block gets Cab[0]
  TrainCount = 0,
  CabH = TrainCount, CabH = SensorH*, TrainCount = SensorH+,
  CabG = TrainCount, CabG = SensorG*, TrainCount = SensorG+,
  CabF = TrainCount, CabF = SensorF*, TrainCount = SensorF+,
  CabE = TrainCount, CabE = SensorE*, TrainCount = SensorE+,
  CabD = TrainCount, CabD = SensorD*, TrainCount = SensorD+,
  CabC = TrainCount, CabC = SensorC*, TrainCount = SensorC+,
  CabB = TrainCount, CabB = SensorB*, TrainCount = SensorB+,
  CabA = TrainCount, CabA = SensorA*, TrainCount = SensorA+,
```

This When-Do assigns each train a cab. The lead train (the train in the "highest" lettered track block) is assigned to Cab[0], the next train to Cab[1], etc.

With that initialization complete, our cab control system is ready to roll. We can simply throttle up our trains using their on-screen pop-up throttles. From then on, each throttle will automatically follow its train around the layout and our built-in collision avoidance system will automatically keep our four trains safely separated.

Using Cab Control with Conventional Throttles:

In this example, we've used the features of CTI's computer-controlled Smart Cab throttle to smoothly start and stop our trains based on traffic conditions ahead. But this same cab control technique can be used with conventional manual throttles as well.

In that case, a simple Train Brain controller can be substituted for each Smart Cab to provide the automated braking function. We simply modify our TCL code to activate the controller (instead of the Smart Cab's brake) to apply the brake and deactivate the controller to release it. Using this technique, we sacrifice the smooth starts and stops provided by the Smart Cab's simulated inertia feature, but functionally things work just the same.

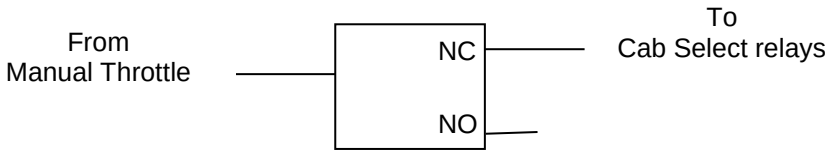

Figure 3. Controller-based brake function

Alternatively, some users may prefer to leave the control of the train completely in the hands of the operator. The computer can be used to handle the automated routing of cabs to follow trains as they move about the layout, leaving the operator free to run his train without worrying about the need to manually route cabs to track blocks. In this case, the operator is fully responsible for obeying trackside signals to avoid collision. He'll receive no help from the PC. For that, only Steps #1, 2b and #3 of the algorithm are required, and the TCL code for our representative block B, is reduced to:

```
When SensorB = True Do
  Wait Until SensorC = False Then
  CabC = CabB, SetRelays(&CabC1, &CabC2, &CabC3)
  EndIf
```

TCL Program Listing:

The complete TCL program for our fully automated four-train operation is shown below. You can simply cut-and-paste it into Tbrain's TCL editor window to give it a try.

This is by no means the only way to solve the cab control problem. Many other implementations are possible. We suggest you use this example as a starting point, and then let your imagination take over. Try adding code for operation in the reverse direction, add code to handle a passing siding, etc. And most of all, *have fun.*

```
Controls: CabA1, CabA2, CabB1, CabB2, CabC1, CabC2, CabD1, CabD2,
          CabE1, CabE2, CabF1, CabF2, CabG1, CabG2, CabH1, CabH2,
          CabA3, CabB3, CabC3, CabD3, CabE3, CabF3, CabG3, CabH3

Sensors:  SensorA#, SensorB#, SensorC#, SensorD#,
          SensorE#, SensorF#, SensorG#, SensorH#

SmartCabs: Cab[4]

Variables: CabA, CabB, CabC, CabD, CabE, CabF, CabG, CabH, TrainCount

Actions:

Sub SetRelays (CabSelect, Relay1, Relay2, Relay3)
  If     CabSelect=0 Then *Relay1=Off, *Relay2=Off, *Relay3=Off
  ElseIf CabSelect=1 Then *Relay1=On,  *Relay2=Off, *Relay3=Off
  ElseIf CabSelect=2 Then *Relay1=Off, *Relay2=Off, *Relay3=On
  ElseIf CabSelect=3 Then *Relay1=Off, *Relay2=On,  *Relay3=On
  EndIf
EndSub

  When $Reset = True Do
    'Initialize cab assignments: engine in highest lettered track block gets Cab[0]
    TrainCount = 0,
    CabH = TrainCount, CabH = SensorH*, TrainCount = SensorH+
    CabG = TrainCount, CabG = SensorG*, TrainCount = SensorG+
    CabF = TrainCount, CabF = SensorF*, TrainCount = SensorF+
    CabE = TrainCount, CabE = SensorE*, TrainCount = SensorE+
    CabD = TrainCount, CabD = SensorD*, TrainCount = SensorD+
    CabC = TrainCount, CabC = SensorC*, TrainCount = SensorC+
    CabB = TrainCount, CabB = SensorB*, TrainCount = SensorB+
    CabA = TrainCount, CabA = SensorA*, TrainCount = SensorA+

    'Configure cab control relays
    SetRelays(CabA, &CabA1, &CabA2, &CabA3)
    SetRelays(CabB, &CabB1, &CabB2, &CabB3)
    SetRelays(CabC, &CabC1, &CabC2, &CabC3)
    SetRelays(CabD, &CabD1, &CabD2, &CabD3)
    SetRelays(CabE, &CabE1, &CabE2, &CabE3)
    SetRelays(CabF, &CabF1, &CabF2, &CabF3)
    SetRelays(CabG, &CabG1, &CabG2, &CabG3)
    SetRelays(CabH, &CabH1, &CabH2, &CabH3)

  When SensorA = True Do
    If CabA = CabH Then Wait Until SensorH = False Then EndIf
    If  SensorB = True Then
      Cab[CabA].Brake = On
      Wait Until SensorB = False Then
      Cab[CabA].Brake = Off
    EndIf
    CabB = CabA, SetRelays(CabB, &CabB1, &CabB2, &CabB3)

  When SensorB = True Do
    If CabB = CabA Then Wait Until SensorA = False Then EndIf
    If SensorC = True Then
      Cab[CabB].Brake = On
```

```
    Wait Until SensorC = False Then
    Cab[CabB].Brake = Off
  EndIf
  CabC = CabB, SetRelays(CabC, &CabC1, &CabC2, &CabC3)

When SensorC = True Do
  If CabC = CabB Then Wait Until SensorB = False Then EndIf
  If SensorD = True Then
    Cab[CabC].Brake = On
    Wait Until SensorD = False Then
    Cab[CabC].Brake = Off
  EndIf
  CabD = CabC, SetRelays(CabD, &CabD1, &CabD2, &CabD3)

When SensorD = True Do
  If CabD = CabC Then Wait Until SensorC = False Then EndIf
  If SensorE = True Then
    Cab[CabD].Brake = On
    Wait Until SensorE = False Then
    Cab[CabD].Brake = Off
  EndIf
  CabE = CabD, SetRelays(CabE, &CabE1, &CabE2, &CabE3)

When SensorE = True Do
  If CabE = CabD Then Wait Until SensorD = False Then EndIf
  If  SensorF = True Then
    Cab[CabE].Brake = On
    Wait Until SensorF = False Then
    Cab[CabE].Brake = Off
  EndIf
  CabF = CabE, SetRelays(CabF, &CabF1, &CabF2, &CabF3)

When SensorF = True Do
  If CabF = CabE Then Wait Until SensorE = False Then EndIf
  If SensorG = True Then
    Cab[CabF].Brake = On
    Wait Until SensorG = False Then
    Cab[CabF].Brake = Off
  EndIf
  CabG = CabF, SetRelays(CabG, &CabG1, &CabG2, &CabG3)

When SensorG = True Do
  If CabG = CabF Then Wait Until SensorF = False Then EndIf
  If SensorH = True Then
    Cab[CabG].Brake = On
    Wait Until SensorH = False Then
    Cab[CabG].Brake = Off
  EndIf
  CabH = CabG, SetRelays(CabH, &CabH1, &CabH2, &CabH3)

When SensorH = True Do
  If CabH = CabG Then Wait Until SensorG = False Then EndIf
  If SensorA = True Then
    Cab[CabH].Brake = On
    Wait Until SensorA = False Then
    Cab[CabH].Brake = Off
  EndIf
  CabA = CabH, SetRelays(CabA, &CabA1, &CabA2, &CabA3)
```