Applications Note: Implementing Fail-Safe Grade Crossings

This applications note examines techniques for implementing automated grade crossings.

At first glance, a grade crossing seems rather straightforward. Just put a sensor on either side of the crossing and write a few lines of TCL code to activate the gate and flashers when the train reaches the first sensor and deactivate them once it passes the second sensor.

However, once we examine the problem in more detail, and endeavor to account for all possible cases (e.g. trains entering from either direction, trains entering the crossing and then backing out, very short vs. very long trains, etc), the problem becomes a bit more daunting.

In the following discussion, we'll examine solutions to each of these complexities.

The Basic Crossing:

Figure 1 shows one method (there are many ways to tackle this problem) of implementing a grade crossing. Here we'll use two sensors: a current sensor to detect the approach of a train well before it reaches the crossing (from either direction), and an IR sensor to detect the safe passage of the entire train past the immediate crossing itself.

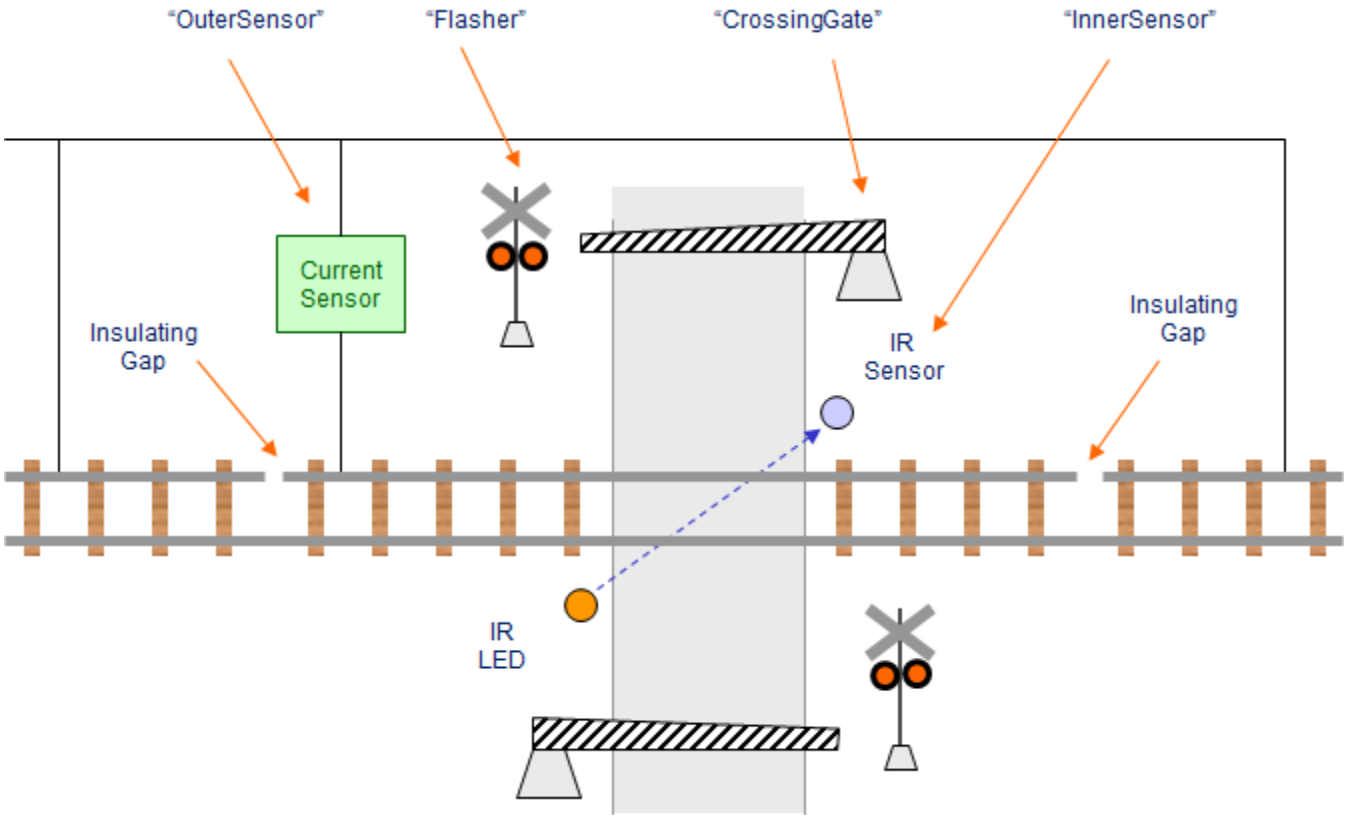

Figure 1. A Basic Grade Crossing

Our current sensor monitors an insulated track block surrounding the crossing. This block should be wide enough to give ample warning of a train's approach from either side. The infrared sensor's emitter/detector pair straddles the crossing diagonally, so that once in the crossing, the IR beam will remain broken until a train has fully cleared in either direction.

With that in mind, here's some TCL code to control our crossing. (We'll assume that the crossing gates are controlled using a Train Brain control relay, and that the alternating flashers on the crossbucks are controlled using two Signalman transistor outputs.)

```
Controls: CrossingGate
Sensors:  OuterSensor#, InnerSensor*~
Signals:  Flasher(2)

Actions:
  When OuterSensor = True Do
    CrossingGate = On
    Wait Until InnerSensor = True Then
      Wait Until InnerSensor = False Then
        CrossingGate = Off

  While CrossingGate = On Do
    Flasher = "*-", Wait 1,
    Flasher = "-*", Wait 1
```

That's all there is to it.

The When-Do statement lowers the crossing gate as soon as the head of an approaching train enters the insulated track block, and is detected by the current sensor "*OuterSensor*". (This works for the approach of a train from either direction.) While the crossing gate is activated, the While-Do statement alternately blinks the crossbuck's two flashers.

Eventually the train reaches the grade crossing itself, and breaks the infrared sensor's beam (setting *InnerSensor* to True). Once the train has completely cleared the crossing, the infrared light beam is re-established, and the value of *InnerSensor* returns to False. At that point our When-Do statement's job is finished, and its final action statement raises the crossing gate. Once that occurs, the While-Do deactivates, and the crossbuck's flashers cease blinking.

Handling Some Pathological Cases:

The implementation of our basic crossing is nearly foolproof. It handles trains of arbitrary length traveling in either direction. However, it fails to handle a few pathological cases.

One situation it isn't equipped to handle occurs when a train enters the region of the crossing, but backs out again on the same side from which it entered before ever reaching the crossing itself. (Such a situation might occur at a crossing in an industrial area where switching operations occur.) In that case, the gate would be activated by the first action statement of our When-Do, but it would then remain lowered forever. Since the train never reached the IR sensor, the condition needed to raise the gate never occurs.

Fortunately, it's a situation that's easy to account for. Consider:

```
Controls: CrossingGate
Sensors:  OuterSensor#, InnerSensor*~
Signals:  Flasher(2)

Actions:
  When OuterSensor = True Do
    CrossingGate = On
    Wait Until InnerSensor = True Or OuterSensor = False Then
      Wait Until InnerSensor = False Then
        CrossingGate = Off

  While CrossingGate = On Do
    Flasher = "*-", Wait 1,
    Flasher = "-*", Wait 1
```

Here we've added a second means to satisfy the condition needed to raise the gate (shown in green in the code above). In this case, if the train backs out of the area before reaching the crossing itself, OuterSensor will return to False. Logically OR'ed with the "normal" condition of `InnerSensor = True`, this pathological condition is detected, allowing execution of the When-Do to proceed, turning off the crossing gate.

A second pathological case that requires a bit of failsafe logic occurs when a train clears the crossing, thereby raising the gate, but then backs into the crossing again before exiting and reentering the insulted track block. In that case, there is no opportunity for the When-Do to execute again to re-lower the crossing gate.

We can add a second When-Do to handle this case.

```
Controls: CrossingGate
Sensors:  OuterSensor#, InnerSensor*~
Signals:  Flasher(2)

Actions:
  When OuterSensor = True Do
    CrossingGate = On
    Wait Until InnerSensor = True Or OuterSensor = False Then
      Wait Until InnerSensor = False Then
        CrossingGate = Off

  When InnerSensor = True Do
    CrossingGate = On
    Wait Until InnerSensor = False Then
      CrossingGate = Off

  While Crossing_Gate = On Do
    Flasher = "*-", Wait 1,
    Flasher = "-*", Wait 1
```

The first When-Do functions exactly as before. The second When-Do will catch a train reentering the crossing any number of times while still in the track block. Under normal operation, its actions are redundant since the gate was already activated when the first When-Do fired. Admittedly, our drivers won't get much warning in this case, but presumably the train will be moving very, very slowly at that point, since it has just reversed direction a few feet away!

With that bit of logic added, our crossing is now 100% foolproof. We leave it to the user to decide if the extra logic to handle the pathological cases is warranted on their layout.