Model Railroad System
A collection of utilities for Model Railroaders
Application Note 02: Two Siding Oval N

Robert Heller
Deepwoods Software
Wendell, MA, USA

January 22, 2015

This documentation was prepared with LaTeX.
This document describes version 2 of the Model Railroad System package.

The author, Robert Heller, may be contacted electronically (E-Mail) via `heller@deepsoft.com`.

Deepwoods Software's web site URL: `http://www.deepsoft.com/`.
ApplicationNote02.tex 1850 2015-01-22 14:53:50Z heller

Contents

Listings

List of Figures

List of Tables

Chapter 1

Introduction

This application note presents the software for a 2' by 6' N scale continious running oval layout. This module is a table top module and features a single track oval with a pair of passing sidings. It is set up to run two trains in *opposite* directions and does so in a fully automated way using computer control, using USB connected IR sensors and control modules from Azatrax. The layout illustrates how to automate signaling and control of trains moving in opposing directions through a single track main line. The ideas presented in this Application Note could be adapted for more realistic layouts and can be incorporated into a CTC system, with automated diversions into passing sidings. This layout, as is, could also be used as a unattended educational museum layout, demostrating how bi-directional traffic is handled on a single track mainline.

This application note expands on information contained in the Programming Guides[2], User Manual[3], and Internals Manual[1]. The *Dispatcher* program, described in the User Manual, was used to create and maintain the dispatcher code described in Chapter 3. This code makes use of the Tcl API for the Azatrax MRD2-U and SR4 devices, which is described in the Internals Manual and Programming Guides. The layout uses some circuits described by Rob Paisley[4].

Chapter 2

The Layout

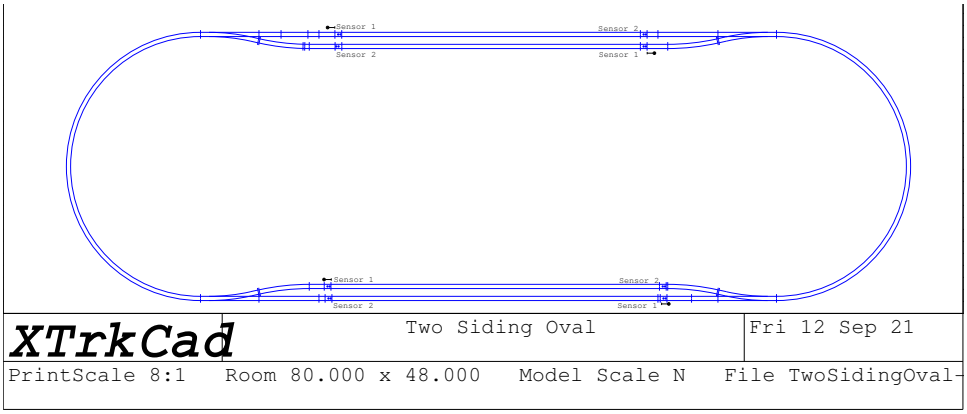

Figure 2.1: Two Siding Oval N

The complete layout is shown in Figure 2.1. This is a basic oval with a pair of passing sidings, one on each straight section. The turnouts at the ends of the passing sidings will be controled by the computer to allow continious running of two trains going in *opposite* directions. If necessary, a train will be held at the end of its siding until the other train clears the single track segment. The computer will use Azatrax MRD2-U IR sensors to sense when the single track segments are occupied and will use Azatrax SR4-U modules to control the direction of travel on the single track segments as well as energizing the twin-coil switch machines to throw the turnouts as needed. The trains will be plain DC powered and the rails will have gaps to isolate the various power sections. The single track segments will be powered from computer controlled reversing relays since these segments

will be operating in alternating directions. The sidings will be fixed polarity wired.

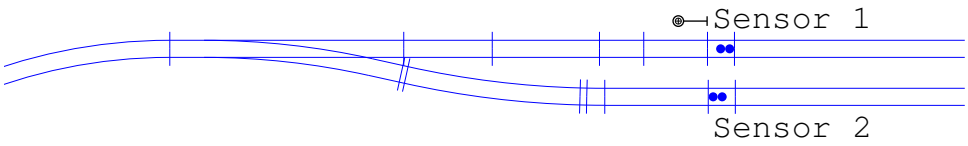

Figure 2.2: Two Siding Oval N, North West Switch with Signal and Sensor locations.

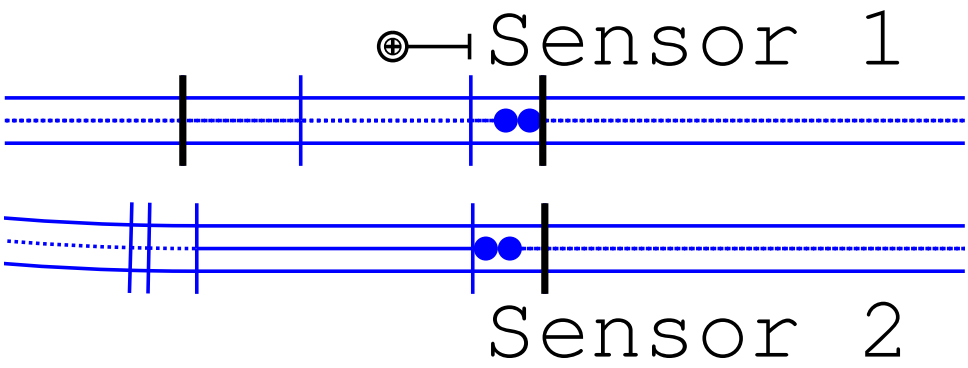

Figure 2.3: Two Siding Oval N, North West Signal and Sensor detail, with gaps.

Figures 2.2 and 2.3 show the detail of the North West Switch with the signal and sensor locations. This is typical for each corner. The inner siding is clockwise running and the outer siding is counter clockwise running[1]. There are eight sensor locations, two for each MRD2-U module. Figure 2.4 shows how the East end sensors are connected. The West end follows a similar pattern. Sensor one for each MRD2-U is the sensor at the entrance end of each logical route through the single track main lines, located at the signal (just before the turnout) and sensor two for that MRD2-U is the sensor at the exit of that logical route through the single track main line (just after the turnout). The end curves handle bi-direction traffic and each MRD2-U device detects trains going in a specific direction. If the software

[1]This happens to correspond to “right hand” running.

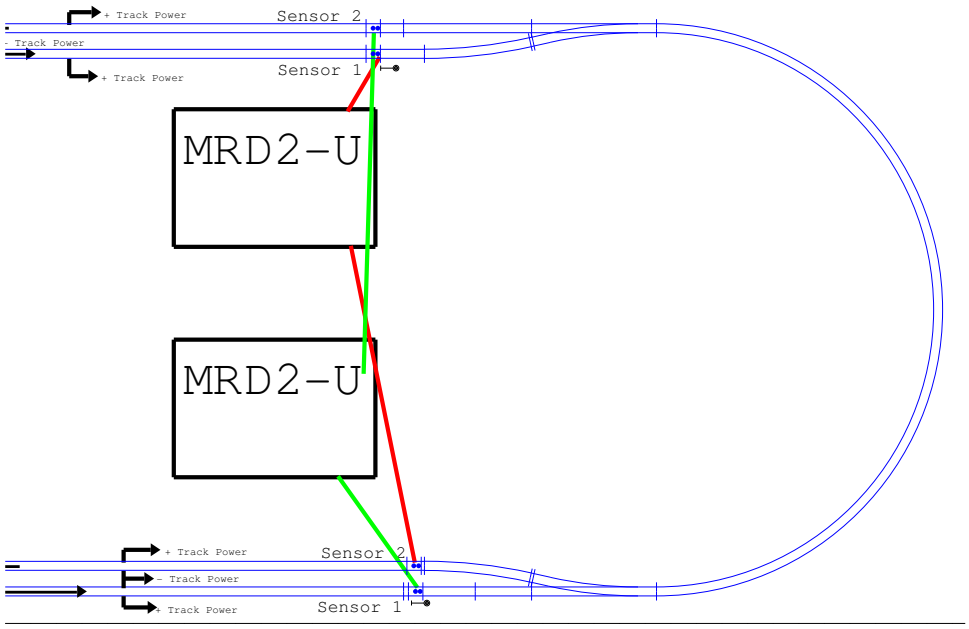

Figure 2.4: Two Siding Oval N, East Sensor connections for MRD2-U modules.

determines that the route is occupied by a train going in one direction, it will hold the train going in the other direction until the first train clears the exit sensor. If the entrance sensor detects a train (which will be stopped at the sensor/signal), the software will change the turnout alignment and flip the direction of travel and will hold opposing traffic.

Chapter 3

Auto Dispatcher

The software for this layout implements a simple automatic dispatching system. Most of the time the trains are on a double track "main line" and require no dispatching, since each main line track segment is operated in a single direction with only one train. At the either end of the layout is a segment of bi-directional single track. Using Azatrax MRD2-U's the computer senses trains traveling through these segments of bi-directional single track and uses Azatrax SR4-Us to control the turnouts, signals, and the polarity of the power feed to the segments of bi-directional single track, allowing only one train access at a time.

Each segment of bi-directional single track, with associated turnouts use two MRD2-U's (one for each logical direction) to sense trains.

3.1 Dispatching Logic

Figure 3.1: Schematic of a bi-direction single track segment

A schematic track diagram of a bi-direction single track segment is shown in Figure 3.1. There are four sensor locations, shown as white dots and labeled `Left 1`, `Left 2`, `Right 1`, and `Right 2`. These sensors are connected to two MRD2-U, `Left` and `Right`. The sensors are connected to the MRD2-U such that a given MRD2-U's "sense 1" is the entry and "sense 2" is the exit.

Each MRD2-U device thus senses a train going in a specific direction through the single track segment. There is a gap (eg with insulating rail joiners) between both mainline tracks and the single track segment. There is an additional gap about the length of the engine past the entrance sensor. This short segment of track is powered via diodes from the rest of the single track segment. The diodes enforce the correct direction of travel for the entering train. The single track segment is considered occupied by a train traveling from right to left if either of the left sensors are active (the train is entering or leaving) or if the left sensor 1 latch is set (the train is between sensors). The single track segment is considered occupied by a train traveling from left to right if either of the right sensors are active (the train is entering or leaving) or if the right sensor 1 latch is set (the train is between sensors). When a train arrives at a sensor 1 position (either `Left 1` or `Right 1`), the computer checks to see if the single track segment is occupied by a train going in the opposite direction.

3.2 The code, annotated.

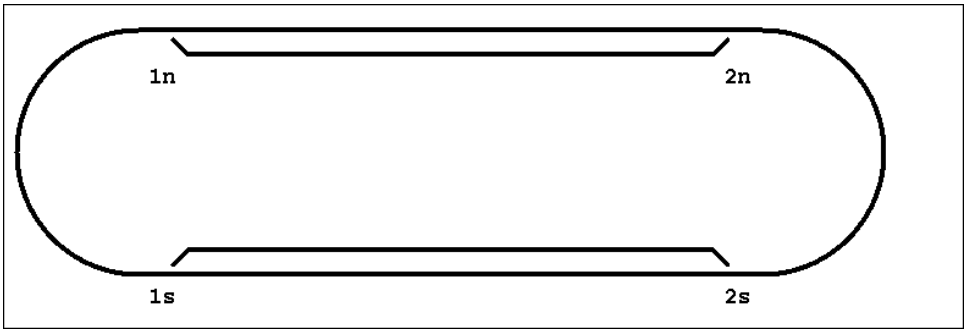

Figure 3.2: Schematic of the layout

Listing 3.1 contains the user code section of the dispatcher program for this layout. The displayed schematic of the layout is shown in Figure 3.2.

The code starts with a procedure named `CurveOccupancy`, starting at line 268 of the source file (`AN02.tcl`). This procedure is used by the single track curved sections at the east and west ends of the layout to check for occupancy. It uses both MRD2-U units to test for either a clockwise or counter-clockwise train occupying the curved section (including the turnouts at the ends).

The next section, starting at line 314, defines a `SNIT` type to handle the logic for occupancy checking for the four straight sections. Four instances of this type are created, one for each of the straight sections, starting at line 431.

Starting at line 446 is a pair of blocks of code to initialize the two SR4 devices to a known powered up state.

At line 460 the main loop starts. The main loop starts by reading in the state data of the six Azatrax devices. This is done at lines 463–473. Then all of the trackwork is "invoked", which runs the occupancy scripts for each piece of trackwork and thus determines whether or not the various blocks are occupied. This is done at lines 475–492.

Next come 4 blocks of code, starting at line 494 that implement the dispatcher logic, which is to check for a train arriving at an entry detector and then testing to see if there is an opposing movement in the single track block. If the block is clear, the turnout is thrown in favor of the entering train and the power reversing relay is set or unset to also favor the entering train. There is a block of code for each entrance sensor. The four blocks of code are at lines 507–523 (West end clockwise block), 525–541 (West end counter-clockwise block), 546–562 (East end clockwise block), and 564–580 (East end counter-clockwise block).

At the bottom of the main loop is a call to the `update`, which causes the GUI to update to reflect the state of the layout.

Listing 3.1: Dispatching logic, implemented in Tcl `AN02.tcl`

```
1
  proc CurveOccupancy {CW_Sensor CCW_Sensor} {
  ## Check for curve occupancy.
  #  First the clockwise sensor is checked and then the counter-
  #  clockwise sensor is checked. The logic is:
6 #  Sense_1 This is at the entrance to the curve, just before the
  #               turnout. If this sensor is active, then a train
  #               is presently entering.
  #  Latch_1 This is the latched state of sense 1. If it is set,
  #               then the train has cleared sense 1, but has not
11 #              yet covered sense 2, this means that the train is
  #               in the curve between the sensors.
  #  Sense_2 This is the sensor at the exit from the curve. If this
  #               sensor is active, then a train is presently
  #               exiting the curve. When this sensor is activated,
16 #              the latch for sensor 1 is cleared.
  #
  #  If all of the above are clear (return false), then the train
  #  has cleared the exit sensor and the curve is not occupied.
```

```
    #  If any of the above are set (return true), then there is a
21  #  train occupying the block.
    #
    #  Both the clockwise and counter-clockwise sensors are checked,
    #  since a tain going in either direction could be occupying the
    #  curve.
26  #
    #  This procedure is used as the Occupancy Script for the curved
    #  sections of the trackwork, including the turnouts.
    #
    #  Parameters:
31  #  CW_Sensor The Clockwise sensor, a MRD object.
    #  CCW_Sensor The Counter-Clockwise sensor, a MRD object.
    #
      if {[$CW_Sensor Sense_1] ||
          [$CW_Sensor Latch_1] ||
36        [$CW_Sensor Sense_2]} {
        return yes
      } elseif {[$CCW_Sensor Sense_1] ||
                [$CCW_Sensor Latch_1] ||
                [$CCW_Sensor Sense_2]} {
41      return yes
      } else {
        return no
      }
    }
46
    snit::type StraightOccupancy {
    ## Object to implement occupancy testing on the straight
    #  sections. This object is used to check occupancy on the
    #  four straight sections.
51  #
    #  Options:
    #  -enter_sense Sensor at the entrance to the straight section.
    #               This will be a sense 2, so only Sense_2 and
    #               Latch_2 are checked.
56  #  -exit_sense Sensor at the exit of the straight section.
    #               This will be a sense 1, so only Sense_1 is
    #               checked.
    #
    #
61  #  A state variable is used to keep track of possible states:
    #  exited, entering, entered, exiting, and default (unknown).
    #  Each state determines which sensors or their latches are
    #  checked and determine the next state.
```

```
#
#  Four of these objects will be created and used in the
#  Occupancy scripts of each of the four straight senctions.
#  The occupiedP method will be called to compute the occupancy
#  state.
#
  option -enter_sense -readonly yes -default {}
  option -exit_sense -readonly yes -default {}
  variable state unknown
  ## State variable.
  constructor {args} {
    ## The constructor just processes the object options.
    $self configurelist $args
  }
  method occupiedP {} {
    ## Method to check for occupancy.  The state variable is
    #  checked and depending on its value, the sensors or latches
    #  are checked to determine the possible progress of a train
    #  through the straight section.
    #
    switch $state {
      exited {
        # A train has exited.  Has a train reached the entry
        # sensor?
        if {[$options(-enter_sense) Sense_2]} {
          # Yes, save the state.
          set state entering
          # Block is now occupied.
          set occupied yes
        } else {
          # No, block is clear.
          set occupied no
        }
      }
      entering {
        # A train was entering.  Has it completly entered yet?
        if {[$options(-enter_sense) Latch_2]} {
          # Yes. The train is now fully in the block.
          set state entered
        }
        # The block is occupied.
        set occupied yes
      }
      entered {
        # A train has completly entered.  Is it leaving yet?
```

```
      if {[$options(-exit_sense) Sense_1]} {
111     # Yes, it is now leaving.
        set state exiting
      }
      # The block is occupied.
      set occupied yes
116   }
      exiting {
        # A train is exiting.  Has it completely left yet?
        if {[$options(-exit_sense) Latch_1]} {
          # Yes, it has now left.
121       set state exited
          # The block is no longer occupied.
          set occupied no
        } else {
          # No, the train has not completely left, so the block
126       # is still occupied.
          set occupied yes
        }
      }
      default {
131     # Unknown state.  Check each possible sensor and
        # determine where the train might be.
        if {[$options(-enter_sense) Sense_2]} {
          # Entry sensor is covered: a train is entering and the
          # block is occupied.
136       set state entering
          set occupied yes
        } elseif {[$options(-enter_sense) Latch_2]} {
          # Entry sensor was covered, but isn't anymore: a train
          # has fully entered and the block is occupied.
141       set state entered
          set occupied yes
        } elseif {[$options(-exit_sense) Sense_1]} {
          # Exit sensor is covered: the train is exiting and the
          # block is occupied.
146       set state exiting
          set occupied yes
        } elseif {[$options(-exit_sense) Latch_1]} {
          # Exit sensor was covered, but isn't anymore: the train
          # has fully exited and the block is no longer occupied.
151       set state exited
          set occupied no
        } else {
          # No sensor state was met.  Presume that the block is
```

```
            # not occupied.
156         set occupied no
          }
        }
      }
      return $occupied
161   }
  }

  # Four occupancy detection objects, one for each of the four
  # straight sections.
166 StraightOccupancy create SouthT1_Occ \
                         -enter_sense WestCounterClockwise \
                         -exit_sense  EastCounterClockwise
  StraightOccupancy create SouthT2_Occ \
                         -enter_sense EastClockwise \
171                      -exit_sense  WestClockwise
  StraightOccupancy create NorthT1_Occ \
                         -enter_sense EastCounterClockwise \
                         -exit_sense  WestCounterClockwise
  StraightOccupancy create NorthT2_Occ \
176                      -enter_sense WestClockwise \
                         -exit_sense  EastClockwise

  # Initialize both Turnout states and both reversing relays.

181 # West end.
  global WestTurnoutState
  WestControl PulseRelays 0 1 0 0 4
  WestControl RelaysOff   0 0 1 0
  set WestTurnoutState normal
186
  # East end.
  global EastTurnoutState
  EastControl PulseRelays 0 1 0 0 4
  EastControl RelaysOff   0 0 1 0
191 set EastTurnoutState normal

  # Main Loop Start
  # The main loop consistes ofthree sections
  while {true} {
196   ## Read all AZATRAX state data: read all sensors from the
    #  devices to the sensor memory buffer.  This data will be
    #  used to check occupicency and to determine if it needful
    #  to throw turnouts and relays.
```

```
201     WestControl GetStateData
        EastClockwise GetStateData
        EastCounterClockwise GetStateData
        EastControl GetStateData
        WestClockwise GetStateData
206     WestCounterClockwise GetStateData

        ## Invoke all trackwork and get occupicency.  Occupicency is
        #  computed from the MRD sensor data loaded above.  Each piece
        #  of trackwork contains an occupicency script which checks the
211     #  sense data and determines if the piece of trackwork is
        #  occupied.

        MainWindow ctcpanel invoke SouthT1
        MainWindow ctcpanel invoke SouthT2
216     MainWindow ctcpanel invoke SE1
        MainWindow ctcpanel invoke Switch1n
        MainWindow ctcpanel invoke SW1
        MainWindow ctcpanel invoke Switch1s
        MainWindow ctcpanel invoke NorthT1
221     MainWindow ctcpanel invoke NorthT2
        MainWindow ctcpanel invoke Switch2n
        MainWindow ctcpanel invoke NE1
        MainWindow ctcpanel invoke NW1
        MainWindow ctcpanel invoke Switch2s
226
        ## Implement dispatcher logic: check for train arrival at the
        #  start of a single track section, then check for a possible
        #  opposing movement. If there is no opposing movement, set the
        #  turnout and reversing relay to favor the newly arrived
231     #  train.
        #
        #  There are four blocks, two for each of two ends.  Each end
        #  has a clockwise and a counter-clockwise block.
        #
236
        ## West end movements.  A clockwise and then counter clockwise
        #  block.

        ## West end clockwise block.
241     # Has a clockwise train arrived at the west end single track
        # segment? If WestClockwise's Sense_1 is covered,
        # WestCounterClockwise sensors are checked to see if there is
        # an opposing movement.
```

```
    if {[WestClockwise Sense_1]} {
246   # Check for opposing movement
      if {![WestCounterClockwise Sense_1] &&
          ![WestCounterClockwise Sense_2] &&
          ![WestCounterClockwise Latch_1]} {
        # single track segment is clear
251     # Throw turnouts 1n and 1s (reversed).
        WestControl PulseRelays 1 0 0 0 4
        WestControl RelaysOn    0 0 1 0;# Set relay.
        set WestTurnoutState reverse
      }
256 }

    ## West end counter-clockwise block.
    # Has a counterclockwise train arrived at the west end single
    # track segment? If WestCounterClockwise's Sense_1 is covered,
261 # WestClockwise sensors are checked to see if there is an
    # opposing movement.
    if {[WestCounterClockwise Sense_1]} {
      # Check for opposing movement
      if {![WestClockwise Sense_1] &&
266       ![WestClockwise Sense_2] &&
          ![WestClockwise Latch_1]} {
        # single track segment is clear
        # Throw turnouts 1n and 1s (normal).
        WestControl PulseRelays 0 1 0 0 4
271     WestControl RelaysOff   0 0 1 0;# Unset relay.
        set WestTurnoutState normal
      }
    }

276 ## East end movements: just like the west end, a clockwise
    #  block and then a counter clockwise block.

    ## East end clockwise block.
    # Has a clockwise train arrived at the east end single track
281 # segment? If EastClockwise's Sense_1 is covered,
    # EastCounterClockwise sensors are checked to see if there
    # is an opposing movement.
    if {[EastClockwise Sense_1]} {
      # Check for opposing movement
286   if {![EastCounterClockwise Sense_1] &&
          ![EastCounterClockwise Sense_2] &&
          ![EastCounterClockwise Latch_1]} {
        # single track segment is clear
```

```
        # Throw turnouts 2n and 2s (normal).
291     EastControl PulseRelays 0 1 0 0 4
        EastControl RelaysOff   0 0 1 0;# Unset relay.
        set EastTurnoutState normal
      }
    }
296
    ## East end counter-clockwise block.
    # Has a counterclockwise train arrived at the east end single
    # track segment? If EastCounterClockwise's Sense_1 is covered,
    # EastClockwise sensors are checked to see if there is an
301 # opposing movement.
    if {[EastCounterClockwise Sense_1]} {
      # Check for opposing movement
      if {![EastClockwise Sense_1] &&
          ![EastClockwise Sense_2] &&
306       ![EastClockwise Latch_1]} {
        # single track segment is clear
        # Throw turnouts 2n and 2s (reverse).
        EastControl PulseRelays 1 0 0 0 4
        EastControl RelaysOn    0 0 1 0;# Set relay.
311     set EastTurnoutState reverse
      }
    }


316 update;# Update display
  }
  # Main Loop End
```

Bibliography

[1] Robert Heller. *Model Railroad System A collection of utilities for Model Railroaders, Internals*, 2007-2013.

[2] Robert Heller. *Model Railroad System A collection of utilities for Model Railroaders, Programming Guides*, 2007-2013.

[3] Robert Heller. *Model Railroad System A collection of utilities for Model Railroaders, User Manual*, 2007-2013.

[4] Rob Paisley. Model railroad & misc. electronics. On the web at the URL: `http://home.cogeco.ca/~rpaisley4/CircuitIndex.html`, 1999.