Model Railroad System

2.2.1

Generated by Doxygen 1.8.17

1 Preface 1

2 Introduction 3

2.1 How this manual is organized. . . . 3

3 Universal Test Program Reference 5

3.1 Main GUI Elements . . . 5

3.1.1 Main Window . . . 5

3.1.2 Open New Port . . . 6

3.2 Tests . . . 6

3.2.1 Test Output Card . . . 6

3.2.2 Wraparound Test . . . 7

4 OpenLCB Program Reference 9

4.1 Start up . . . 9

4.1.1 Command Line Options . . . 9

4.1.2 GUI Startup . . . 10

4.2 Main GUI Elements . . . 10

4.2.1 Configuration Tools . . . 12

4.2.1.1 Memory Configuration Options . . . 12

4.2.1.2 Configuration R/W Tool . . . 13

4.2.1.3 CDI Configuration Tool . . . 13

4.2.2 Event Tools . . . 14

4.2.2.1 Send Event Tool . . . 14

4.2.2.2 Received Events . . . 15

5 OpenLCB Daemons (Hubs and Virtual nodes) 17

5.1 Hub Daemons . . . 17

5.2 Virtual Nodes . . . 18

5.2.1 Common Node Configuration . . . 19

5.2.2 EventExchange node for Azatrax MRD2 boards. . . . 19

5.2.2.1 XML Schema for configuration files . . . 20

5.2.3 EventExchange node for Raspberry Pi GPIO pins. . . . 21

5.2.3.1 XML Schema for configuration files . . . 21

5.2.4 EventExchange node for MCP23008 GPIO pins. . . . 22

5.2.4.1 XML Schema for configuration files . . . 23

5.2.5 EventExchange node for MCP23017 GPIO pins. . . . 24

5.2.5.1 XML Schema for configuration files . . . 24

5.2.6 EventExchange node for MCP23017 as signal heads. . . . 25

5.2.6.1 XML Schema for configuration files . . . 26

5.2.7 EventExchange node for the quad signal head HAT. . . . 27

5.2.7.1 XML Schema for configuration files . . . 27
5.2.8 EventExchange node for a SPI connected MAX7221 Signal Driver. . . . 28
5.2.8.1 XML Schema for configuration files . . . 28
5.2.9 EventExchange node for virtual track circuits. . . . 29
5.2.9.1 XML Schema for configuration files . . . 30
5.2.10 EventExchange node for logic blocks. . . . 32
5.2.10.1 XML Schema for configuration files . . . 32
5.2.11 EventExchange node for a CTI Acela network. . . . 33
5.2.11.1 XML Schema for configuration files . . . 35
5.2.12 EventExchange node for a C/MRI network. . . . 36
5.2.12.1 XML Schema for configuration files . . . 37

6 Offline LCC Node Editor Reference **39**
6.1 Command Line Parameters and Options . . . 39
6.1.1 Options . . . 39
6.1.2 X11 Resource Options . . . 39
6.1.2.1 Other options . . . 39
6.1.3 Parameters . . . 40
6.2 Main GUI Elements . . . 40

7 Layout Control Database **41**
7.1 Turnouts . . . 41
7.2 Blocks . . . 41
7.3 Signals . . . 42
7.4 Sensors . . . 42
7.5 Controls . . . 42

8 Azatrax Test Programs Reference **43**
8.1 MRD Test Program Reference . . . 43
8.1.1 Synopsis . . . 43
8.1.2 Main GUI Screen . . . 43
8.2 MRD Sensor Loop Reference . . . 44
8.2.1 Synopsis . . . 44
8.2.2 Main GUI Screen . . . 45
8.3 SR4 Test Program Reference . . . 45
8.3.1 Synopsis . . . 45
8.3.2 Main GUI Screen . . . 45
8.4 SL2 Test Program Reference . . . 46
8.4.1 Synopsis . . . 46
8.4.2 Main GUI Screen . . . 46

8.5 Azatrax Device Map Reference . . . 47
8.5.1 Synopsis . . . 47
8.5.2 Main GUI Screen . . . 47

9 XPressNet Throttle **49**
9.1 Main GUI . . . 49
9.2 Programming Mode . . . 50
9.3 Open Port . . . 50

10 Generic Throttle **53**
10.1 Main GUI . . . 53
10.2 Programming Mode . . . 54

11 Time Table (V2) Tutorial **55**
11.1 Creating a new time table . . . 55
11.1.1 Creating stations . . . 56
11.1.2 Creating cabs . . . 56
11.2 Creating trains . . . 56
11.3 Printing a time table . . . 57

12 Time Table (V2) Reference **59**
12.1 Command Line Usage . . . 59
12.2 Layout of the Main GUI . . . 59
12.3 Creating a New Time Table . . . 61
12.3.1 Creating the station stops for a new time table . . . 64
12.3.2 Create All Cabs Dialog . . . 64
12.4 Loading an Exiting Time Table File . . . 65
12.5 Saving a Time Table File . . . 66
12.6 Adding Trains . . . 67
12.6.1 Create New Train Dialog . . . 68
12.7 Deleting Trains . . . 71
12.8 Linking and Unlinking Duplicate Stations . . . 72
12.9 Adding Station Storage Tracks . . . 73
12.10 Adding Cabs . . . 74
12.11 Handling Notes . . . 75
12.11.1 Creating New Notes and Editing Existing Notes . . . 75
12.11.2 Adding and Removing a Notes To Trains . . . 78
12.12 Printing a Time Table . . . 81
12.12.1 Print Timetable Dialog . . . 82
12.12.2 Print Configuration Dialog . . . 83
12.13 Exiting From the Program . . . 86

12.14 Select One Train Dialog . . . 87
12.15 The View Menu . . . 88
12.15.1 Trains . . . 88
12.15.2 Stations . . . 88
12.15.3 Notes . . . 89
12.16 System Configuration . . . 89
12.17 Add Cab Dialog . . . 89
12.18 Add Remove Note Dialog . . . 89
12.19 Edit Note Dialog . . . 89
12.20 Edit System Configuration . . . 89
12.21 Edit Train Dialog . . . 89
12.22 Select A Storage Track Name . . . 89
12.23 Select One Note Dialog . . . 89
12.24 Select One Station Dialog . . . 89

13 Freight Car Forwarder (V2) Tutorial **91**
13.1 Loading System Data . . . 91
13.2 Assigning Cars . . . 91
13.3 Running Trains . . . 92
13.4 Printing Yard and Switch Lists . . . 92
13.5 Saving the updated car data . . . 92
13.6 Generating Reports . . . 92
13.7 Other activities . . . 92

14 Freight Car Forwarder (V2) Reference **93**
14.1 Command Line Usage . . . 93
14.2 Layout of the Main GUI . . . 93
14.3 Opening and loading a system file. . . . 96
14.4 Loading and reloading the cars file. . . . 97
14.5 Saving the cars file. . . . 98
14.6 Managing trains and printing . . . 99
14.6.1 Controlling Yard Lists . . . 101
14.6.2 Enabling printing for all trains . . . 102
14.6.3 Disabling printing for all trains . . . 102
14.6.4 Printing a dispatcher report . . . 102
14.6.5 Listing local trains for this shift . . . 102
14.6.6 Listing manifests for this shift . . . 102
14.6.7 Listing all trains for all shifts . . . 103
14.6.8 Managing one train . . . 103
14.7 Viewing a car's information . . . 103

14.8 Editing a car's information . . . 105
14.9 Adding a new car . . . 107
14.10 Deleting an existing car . . . 108
14.11 Showing cars without assignments . . . 109
14.12 Running the car assignment procedure . . . 110
14.13 Running every train in the operating session . . . 111
14.14 Running the box move trains . . . 113
14.15 Running a single train . . . 114
14.16 Opening a Printer . . . 115
14.17 Closing the printer . . . 118
14.18 Printing yard and switch lists . . . 119
14.19 Showing cars on the screen . . . 120
14.20 Printing Reports . . . 122
14.21 Resetting Industry Statistics . . . 124
14.22 Quiting the application . . . 125
14.23 General Dialogs . . . 126
14.23.1 Control Yard Lists Dialog . . . 126
14.23.2 Enter Owner Initials Dialog . . . 126
14.23.3 Select A Train Dialog . . . 126
14.23.4 Manage One Train Dialog . . . 127
14.23.5 Open Printer Dialog . . . 127
14.23.6 Search For Cars Dialog . . . 127
14.23.7 Select A Division Dialog . . . 128
14.23.8 Select An Industry Dialog . . . 128
14.23.9 Select A Station Dialog . . . 128
14.23.10 Select Car Type . . . 128
14.24 Data files . . . 128
14.24.1 Data File Formats . . . 129
14.24.1.1 System File . . . 129
14.24.1.2 Industry File . . . 130
14.24.1.3 Trains File . . . 131
14.24.1.4 Orders File . . . 131
14.24.1.5 Owners File . . . 131
14.24.1.6 Car Types File . . . 131
14.24.1.7 Cars File . . . 132
14.24.1.8 Statistics File . . . 132
14.24.1.9 Other data files . . . 132

15 Resistor Program Reference **133**

16 LocoPull Program Reference **135**
16.1 Basis and Mathematics . . . 135
16.2 The GUI . . . 136
16.2.1 The Scale . . . 137
16.2.2 Locomotive Information . . . 138
16.2.3 Consist Information . . . 138
16.2.4 Zero-grade capacity . . . 138
16.2.5 Grade information . . . 138
16.2.6 Curve information . . . 138
16.2.7 Capacity and Grade plus Curve . . . 138

17 Camera Programs Reference **139**

18 Dispatcher Tutorial **141**
18.1 A "Simple Mode" CTC Panel . . . 141
18.2 A LCC Example . . . 148

19 Dispatcher Reference **167**
19.1 Main GUI Screen . . . 167
19.1.1 Track work Node Graphs . . . 168
19.1.1.1 Loading a Layout . . . 169
19.1.1.2 Finding Nodes . . . 169
19.1.1.3 Printing Node Graphs . . . 169
19.1.2 Creating a new CTC Panel . . . 169
19.1.3 Opening an existing CTC Panel . . . 170
19.2 Configurable Options . . . 170
19.3 CTC Panel Windows . . . 171
19.3.1 Menu items available when editing a CTC Panel Window . . . 171
19.3.1.1 File menu . . . 171
19.3.1.2 Edit menu . . . 172
19.3.1.3 View menu . . . 172
19.3.1.4 Panel menu . . . 172
19.3.1.5 C/Mri menu . . . 173
19.3.1.6 Azatrax menu . . . 173
19.4 CTC Panel Code . . . 173
19.4.1 Wrapped CTC Panel Programs . . . 173
19.4.2 Generated Code . . . 173
19.4.2.1 Configuring CTC Panel Windows . . . 174
19.4.2.2 Adding, Editing, and deleting elements to CTC Panel Windows . . . 174
19.4.2.3 Adding, Editing, and deleting C/Mri nodes to CTC Pane Windows . . . 175

19.4.2.4 Adding, Editing, and deleting Azatrax nodes to CTC Panel Windows . . . 175
19.4.3 User Code . . . 175
19.4.3.1 Insert-able Modules . . . 175
19.4.3.2 The Main Loop . . . 175
19.5 Add CMRI Node Dialog . . . 176
19.6 Select CMRI Node Dialog . . . 176
19.7 Add Azatrax Node Dialog . . . 177
19.8 Select Azatrax Node Dialog . . . 178
19.9 Add Panel Object Dialog . . . 179
19.10 Select Panel Object Dialog . . . 180
19.11 Edit User Code Dialog . . . 181
19.12 Find Node Dialog . . . 182
19.13 Print Dialog . . . 182
19.14 Select Panel Dialog . . . 183
19.15 Using the Dispatcher program with layouts designed in XtrackCAD . . . 184
19.15.1 LCC event id format. . . . 184
19.15.2 XTrackCAD "script" formats. . . . 184
19.15.3 Layout Controls Dialog . . . 184
19.16 Insertable Module Library . . . 184
19.16.1 Track Work type . . . 184
19.16.1.1 Blocks::Block . . . 185
19.16.1.2 Switches::Switch . . . 185
19.16.2 Switch Plate type . . . 185
19.16.3 Signal types . . . 185
19.16.4 Signal Plate type . . . 186
19.16.5 Control Point type . . . 186
19.16.6 Radio Group type . . . 186

20 Dispatcher Examples **187**
20.1 Example 1: Simple siding on single track mainline . . . 187
20.2 Example 2: Mainline with an industrial siding . . . 193
20.3 Example 3: double track crossover . . . 204
20.4 Example 4: From Chapter 9 of C/MRI User's Manual V3.0 . . . 213

21 SatelliteDaemon **221**
21.1 SYNOPSIS . . . 221
21.2 DESCRIPTION . . . 221
21.3 OPTIONS . . . 221
21.4 PROTOCOL . . . 222
21.5 AUTHOR . . . 222

22 raildriverd **223**
22.1 SYNOPSIS . . . 223
22.2 DESCRIPTION . . . 223
22.3 OPTIONS . . . 223
22.4 PARAMETERS . . . 223
22.5 Hotplugging scripts and setup. . . . 224
22.6 AUTHOR . . . 224

23 OpenLCB Tcp/Ip Hub Server **225**
23.1 SYNOPSIS . . . 225
23.2 DESCRIPTION . . . 225
23.3 PARAMETERS . . . 225
23.4 OPTIONS . . . 225
23.5 AUTHOR . . . 226

24 OpenLCB GridConnect Tcp/Ip Hub Server **227**
24.1 SYNOPSIS . . . 227
24.2 DESCRIPTION . . . 227
24.3 PARAMETERS . . . 227
24.4 OPTIONS . . . 227
24.5 AUTHOR . . . 228

25 OpenLCB Router Daemon (Server) **229**
25.1 SYNOPSIS . . . 229
25.2 DESCRIPTION . . . 229
25.3 PARAMETERS . . . 229
25.4 OPTIONS . . . 229
25.5 AUTHOR . . . 230

26 OpenLCB MRD2 Node **231**
26.1 SYNOPSIS . . . 231
26.2 DESCRIPTION . . . 231
26.3 PARAMETERS . . . 231
26.4 OPTIONS . . . 231
26.5 CONFIGURATION . . . 232
26.6 AUTHOR . . . 232

27 OpenLCB PiGPIO node **233**
27.1 SYNOPSIS . . . 233
27.2 DESCRIPTION . . . 233
27.3 PARAMETERS . . . 233

27.4 OPTIONS . . . 233
27.5 CONFIGURATION . . . 234
27.6 AUTHOR . . . 234

28 OpenLCB PiMCP23008 node **235**
28.1 SYNOPSIS . . . 235
28.2 DESCRIPTION . . . 235
28.3 PARAMETERS . . . 235
28.4 OPTIONS . . . 235
28.5 CONFIGURATION . . . 236
28.6 AUTHOR . . . 236

29 OpenLCB PiMCP23017 node **237**
29.1 SYNOPSIS . . . 237
29.2 DESCRIPTION . . . 237
29.3 PARAMETERS . . . 237
29.4 OPTIONS . . . 237
29.5 CONFIGURATION . . . 238
29.6 AUTHOR . . . 238

30 OpenLCB PiMCP23017 as signal driver node **239**
30.1 SYNOPSIS . . . 239
30.2 DESCRIPTION . . . 239
30.3 PARAMETERS . . . 239
30.4 OPTIONS . . . 239
30.5 CONFIGURATION . . . 240
30.6 AUTHOR . . . 240

31 OpenLCB program for the MCP23017-based quad signal head HAT **241**
31.1 SYNOPSIS . . . 241
31.2 DESCRIPTION . . . 241
31.3 PARAMETERS . . . 241
31.4 OPTIONS . . . 241
31.5 CONFIGURATION . . . 242
31.6 AUTHOR . . . 242

32 OpenLCB PiSPIMax7221 node **243**
32.1 SYNOPSIS . . . 243
32.2 DESCRIPTION . . . 243
32.3 PARAMETERS . . . 243
32.4 OPTIONS . . . 243

32.5 CONFIGURATION . . . 244
32.6 AUTHOR . . . 244

33 OpenLCB Virtual Track Circuits node **245**
33.1 SYNOPSIS . . . 245
33.2 DESCRIPTION . . . 245
33.2.1 Code rate and aspect. . . . 245
33.3 PARAMETERS . . . 246
33.4 OPTIONS . . . 246
33.5 CONFIGURATION . . . 246
33.6 AUTHOR . . . 246

34 OpenLCB Logic node **247**
34.1 SYNOPSIS . . . 247
34.2 DESCRIPTION . . . 247
34.3 PARAMETERS . . . 247
34.4 OPTIONS . . . 248
34.5 CONFIGURATION . . . 248
34.6 AUTHOR . . . 248

35 OpenLCB Acela Node **249**
35.1 SYNOPSIS . . . 249
35.2 DESCRIPTION . . . 249
35.3 PARAMETERS . . . 249
35.4 OPTIONS . . . 249
35.5 CONFIGURATION . . . 250
35.6 AUTHOR . . . 250

36 OpenLCB CMRI Node **251**
36.1 SYNOPSIS . . . 251
36.2 DESCRIPTION . . . 251
36.3 PARAMETERS . . . 251
36.4 OPTIONS . . . 251
36.5 CONFIGURATION . . . 252
36.6 AUTHOR . . . 252

37 JMRI Tables to LayoutDB converter **253**
37.1 SYNOPSIS . . . 253
37.2 DESCRIPTION . . . 253
37.3 PARAMETERS . . . 253
37.4 OPTIONS . . . 253

37.5 AUTHOR . . . 253

38 LayoutDB to JMRI Tables converter **255**

38.1 SYNOPSIS . . . 255
38.2 DESCRIPTION . . . 255
38.3 PARAMETERS . . . 255
38.4 OPTIONS . . . 255
38.5 AUTHOR . . . 255

39 Offline LCC Node Editor **257**

39.1 SYNOPSIS . . . 257
39.2 DESCRIPTION . . . 257
39.3 PARAMETERS . . . 257
39.4 OPTIONS . . . 258
39.4.1 X11 Resource Options . . . 258
39.4.2 Other options . . . 258
39.5 AUTHOR . . . 258

40 Help **259**

41 Version **261**

42 GNU GENERAL PUBLIC LICENSE **263**

Bibliography **269**

Index **271**

Chapter 1

Preface

This is the user manual for the Model Railroad system. It is a "work in progress" and I will be adding chapters as I write the various self-contained "main programs".

Chapter 2

Introduction

2.1 How this manual is organized.

This manual is broken up into chapters, one or two for each "main program". [1] These chapters are:

- Chapter Universal Test Program Reference documents the Universal Test program. This program is part of the CMR/I (Chubb) library and is used to test CMR/I nodes.
- Chapter OpenLCB Program Reference documents the OpenLCB program. This program implements a dianostic program for LCC networks, and includes an event logger, event injector, and memory configuration tools (both raw hex and CDI).
- Chapter OpenLCB Daemons (Hubs and Virtual nodes) documents the OpenLCB daemon programs. This includes both the Hub Daemons and the Virtual Node Daemons. The Hub Daemons implement virtual networks over Tcp/Ip. The Virtual Node Daemons implement LCC nodes as processes running on a host computer.
- Chapter Azatrax Test Programs Reference documents the Azatrax Test programs. These programs are part of the Azatrax library are are used to test Azatrax USB modules
- Chapter XPressNet Throttle documents the XPressNet Throttle program. This program is part of the XPressNet library and is a simple GUI program that implements the functionallity of the DCC control unit (aka a throttle).
- Chapter Generic Throttle documents the Generic Throttle program. This program implements the functionallity of a generic throttle. It can be used as the basis for a DC or DCC control unit or throttle, using a specific control library.
- Chapters Time Table (V2) Tutorial and Time Table (V2) Reference document the Time Table (V2) program. This program is used to create employee timetables.
- Chapters Freight Car Forwarder (V2) Tutorial and Freight Car Forwarder (V2) Reference document the Freight Car Forwarder (V2) program. This program is used to create switch lists for freight car forwarding.
- Chapters Resistor Program Reference and LocoPull Program Reference document the calculator scripts, Resistor and LocoPull, that are available to help model railroaders perform some common calculations.
- Chapter Camera Programs Reference documents the camera scripts. These scripts perform various camera scene calculations that are useful for model railroaders.

[1] The various programming libraries are described in the programming guides[4].

- Chapters Dispatcher Tutorial, Dispatcher Reference, and Dispatcher Examples document the automated dispatcher program.
- Chapter SatelliteDaemon documents the daemon for using satellite computers.
- Chapter raildriverd documents the daemon program for the RailDriver control stand console.
- Chapter OpenLCB Tcp/Ip Hub Server documents the daemon program for the binary OpenLCB over Tcp Hub.
- Chapter OpenLCB GridConnect Tcp/Ip Hub Server documents the daemon program for the OpenLCB Grid↩ Connect over Tcp Hub.
- Chapter OpenLCB MRD2 Node documents the daemon program for the OpenLCB interface to the Azatrax MRD2 USB connected IR detectors.
- Chapter OpenLCB PiGPIO node documents the daemon program for the OpenLCB interface to the Raspberry PI's GPIO pins.
- Chapter OpenLCB Virtual Track Circuits node documents the daemon program for the OpenLCB Virtual Track Circuits.
- Chapter OpenLCB Logic node documents the daemon program for the OpenLCB Logic module.
- Chapter OpenLCB Acela Node documents the daemon program for the CTI Acela.

Chapter 3

Universal Test Program Reference

The Universal Test program is used to test the I/O ports on a USIC, SUSIC, or SMINI node. It is a port of the universal test program that is shown in [2] and [3].

3.1 Main GUI Elements

3.1.1 Main Window

The main window upon start up looks like this:

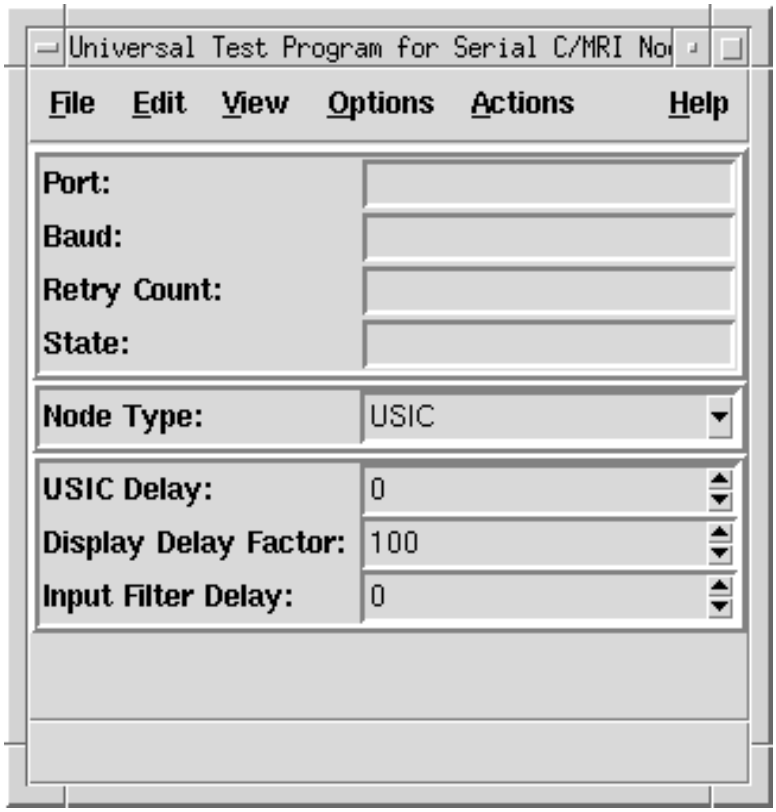


Figure 3.1 The main window of the Universal Test Program

The node type and initialization factors can be set. The Display Delay Factor is the number of hundredths of seconds between bit tests. The default value of 100 means a 1 second delay between output bits. The Input Filter Delay is the number of hundredths of seconds between bit tests for the input port (wraparound) test. The default value of 0 means to test as fast as possible (the program will stop if there is an error).

3.1.2 Open New Port

The New menu item on the File menu opens the serial port (/dev/ttySn) the Chubb node is connected to and set the baud rate and retry count, as shown here:

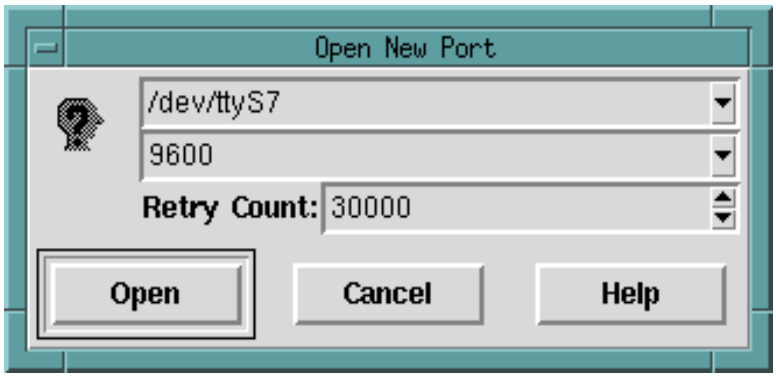


Figure 3.2 The Open New Port dialog box of the Universal Test Program

The Open menu item on the File menu opens the previously open port (if the port is currently open, it is closed first). The board at UA 0 is then initialized. For USIC and SUSIC cards, it is presumed that the backplane contains just one output card (in the first slot) for output testing, and one each output and input card for the wraparound test (output card in the first slot and the input card in the second slot).

3.2 Tests

There are two tests available: the output port test, which tests an output port card and the wraparound test, which tests an input port card using an output port card. The output card test uses an output card LED test plug in and lights up one LED at a time. The wraparound uses the wraparound cable to connect an input card to an output card and writes bit values to the output card and reads these values on the input card and compares what was written with what was read. These tests are selected from the Actions menu.

3.2.1 Test Output Card

The output card test displays the dialog box shown here:

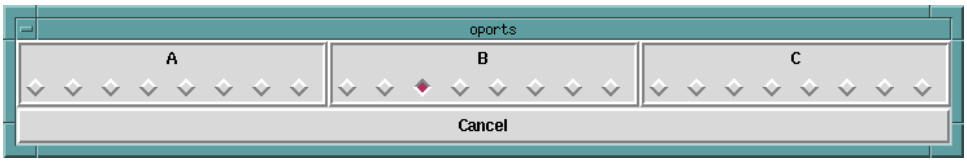


Figure 3.3 The Output Card Test Dialog Box

The lit up indicator on the dialog box should match a corresponding LED on the output card LED test plug. The test is repeated until canceled.

3.2.2 Wraparound Test

The wraparound test displays the dialog box shown here:

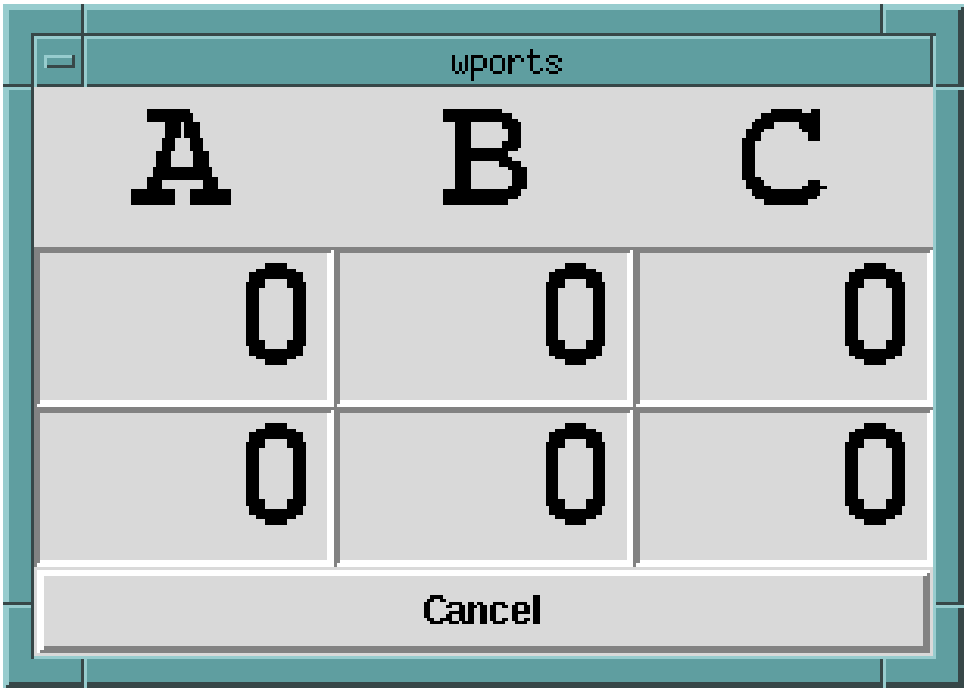


Figure 3.4 The Wraparound Test Dialog Box

The hexadecimal numbers represent the bit pattern written to the output card. These values are read back from the input card and compared. If there is a difference, the test stops and displays the error bit pattern.

Chapter 4

OpenLCB Program Reference

The OpenLCB Program is used for configuring OpenLCB nodes and for testing an OpenLCB network. It can also manage a Layout Control Database, which can be used by the Offline LCC Node Editor and Dispatcher (see Dispatcher Reference manual) programs.

4.1 Start up

When the OpenLCB program starts it connects as a OpenLCB node to a network of OpenLCB nodes. This network could be over a CAN bus network or it could be over an Ethernet network using Tcp/Ip, or using some other form of networked interconnection. It could also use different interconnection network technologies.

4.1.1 Command Line Options

The OpenLCB program takes some command line options that define how it will connect to other OpenLCB nodes. The first thing that is needed is the constructor class for the transport layer that will be used. This is the layer that is the software driver for the specific wire protocol layer that will be used. Next are the I/O options used by that constructor. Usually this is the name of the I/O device or other information needed to connect to the device. It might also include any additional connection information like the node identification.

These command line options are:

- -transportname The name of the transport constructor. A shell wildcard is allowed (but needs to be quoted or escaped).
- -listconstructors Print a list of available constructors and exit.
- -help Print a short help message and exit.

Additional options, specific to the transport constructor can also be specified.

4.1.2 GUI Startup

If the command line options are not specified or not fully specified, the program uses dialog boxes to gather the necessary options it needs to connect. The first dialog box selects the transport constructor to use. It looks like this:

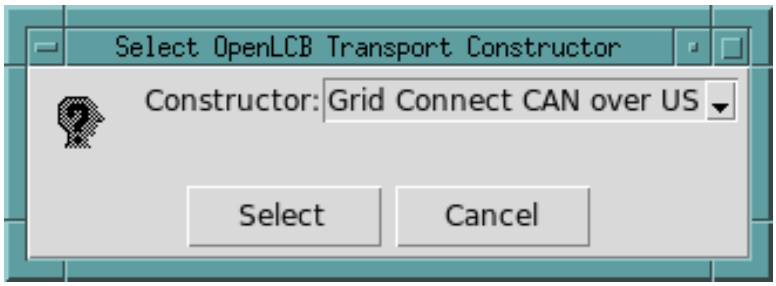


Figure 4.1 OpenLCB Transport Selection Dialog

This dialog box contains a drop down menu of the available (known) transport layer constructors, along with buttons to to either select the transport constructor or to cancel the process.

After selecting the transport constructor, the options for the transport constructor are selected with a constructor specific dialog box. The dialog box for the Grid Connect CAN over USBSerial one looks like this:

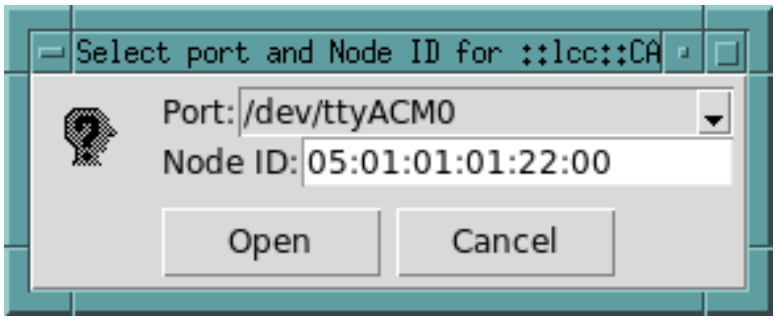


Figure 4.2 Grid Connect CAN over USBSerial Options Dialog

This dialog box selects the serial port device name and the Node ID to use for this connection. There are buttons to open the connection or to cancel the operation.

Once the transport constructor and its options are selected the program starts and displays the main window.

4.2 Main GUI Elements

The main window of the application contains a list of nodes on the network(s) it is connected to. This looks like this:

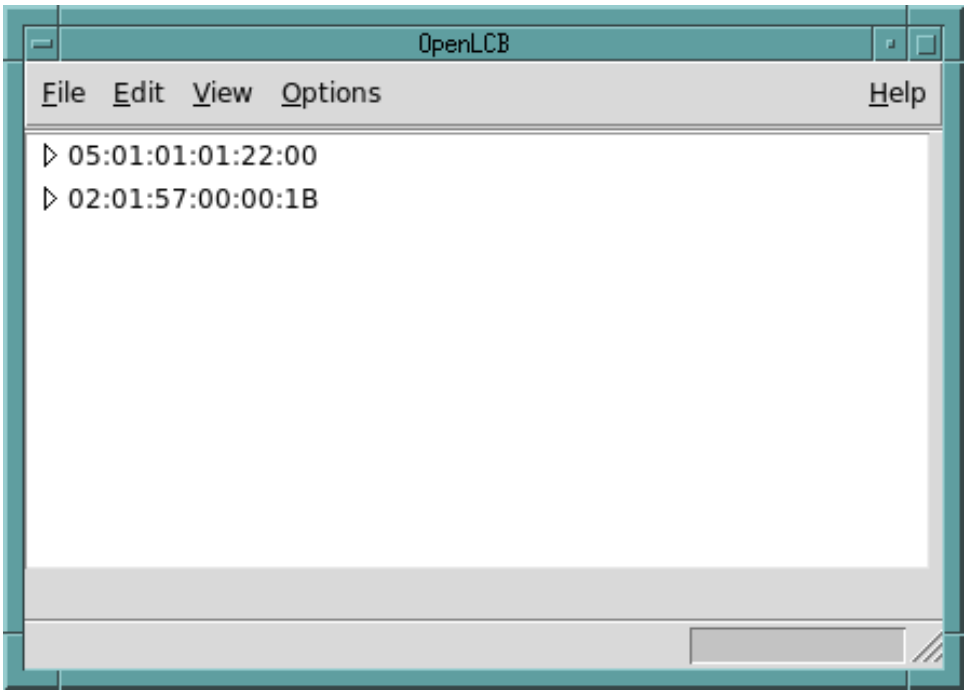


Figure 4.3 OpenLCB Main Window, with the node trees closed

Each node is listed by Node ID. The node trees can be opened to reveal both the simple node information as well as the supported protocols, as shown here:

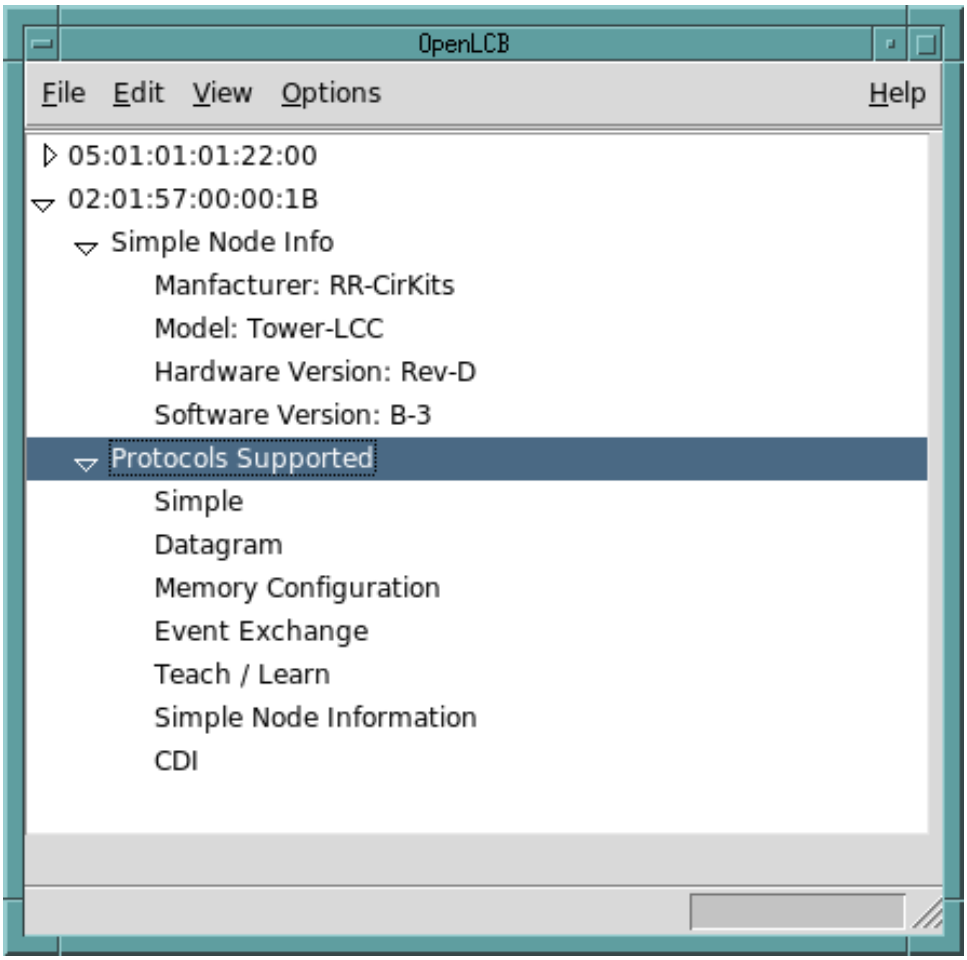


Figure 4.4 OpenLCB Main Window, with the node trees opened

The node information tree contains leaf nodes containing information about the node, including the name of its manufacturer, its model, and it hardware and software (firmware) version numbers. Sometimes nodes can be assigned user supplied names and descriptions. This information is also displayed.

The Memory Configuration and CDI protocol items can be clicked to open up configuration tools.

4.2.1 Configuration Tools

There are two configuration tools available. A simple memory read/write tool and a structured configuration tool that uses a GUI generated from the CDI information supplied by the node itself.

4.2.1.1 Memory Configuration Options

The simple memory read/write tool provides a map of what sorts of memory is available to be configured. This dialog box looks like this:

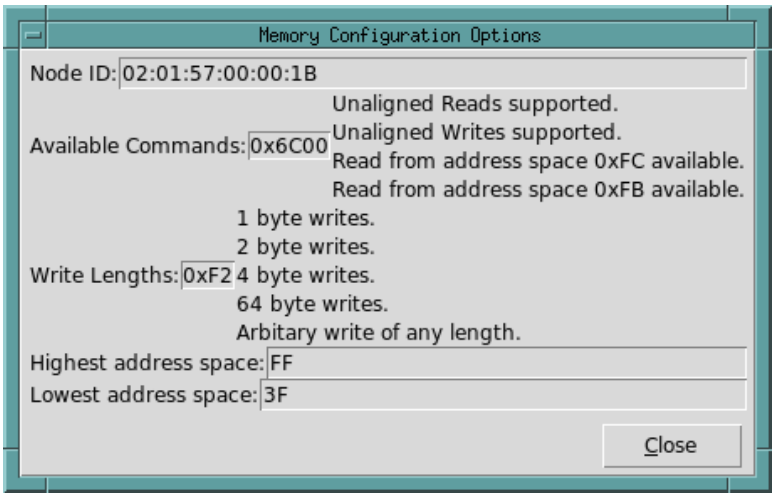


Figure 4.5 Memory Configuration Options Display Dialog Box

This dialog box displays the available commands bitmap, both in hex and with the textual description of the on bits. It does the same for the write lengths. It also shows the highest and lowest memory spaces and if there is a name, it displays that too.

4.2.1.2 Configuration R/W Tool

The simple memory read/write tool is just a simple tool that reads and writes a block of up to 64 bytes of memory. The tool looks like this:

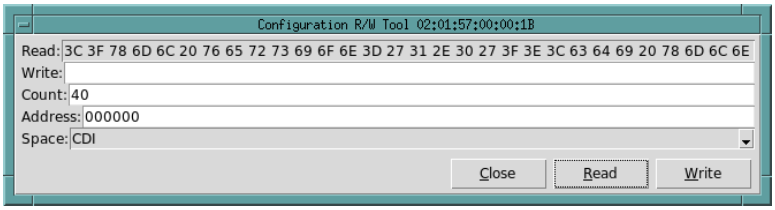


Figure 4.6 Memory Read/Write Configuration tool

This dialog box displays a block of memory read back as pairs of hex digits. It has an entry area to enter sequence of bytes in hex to be written to the node's memory. There is a space for the count, the starting address, and a drop down menu of possible spaces to read from or write to. There are buttons to close, read, or write at the bottom.

4.2.1.3 CDI Configuration Tool

The other memory configuration tool uses the node supplied XML coded CDI to define the structure of the node's configuration memory. It creates a node specific configuration window. Here is the one created for a RR-Cirkits Tower-LLC node:

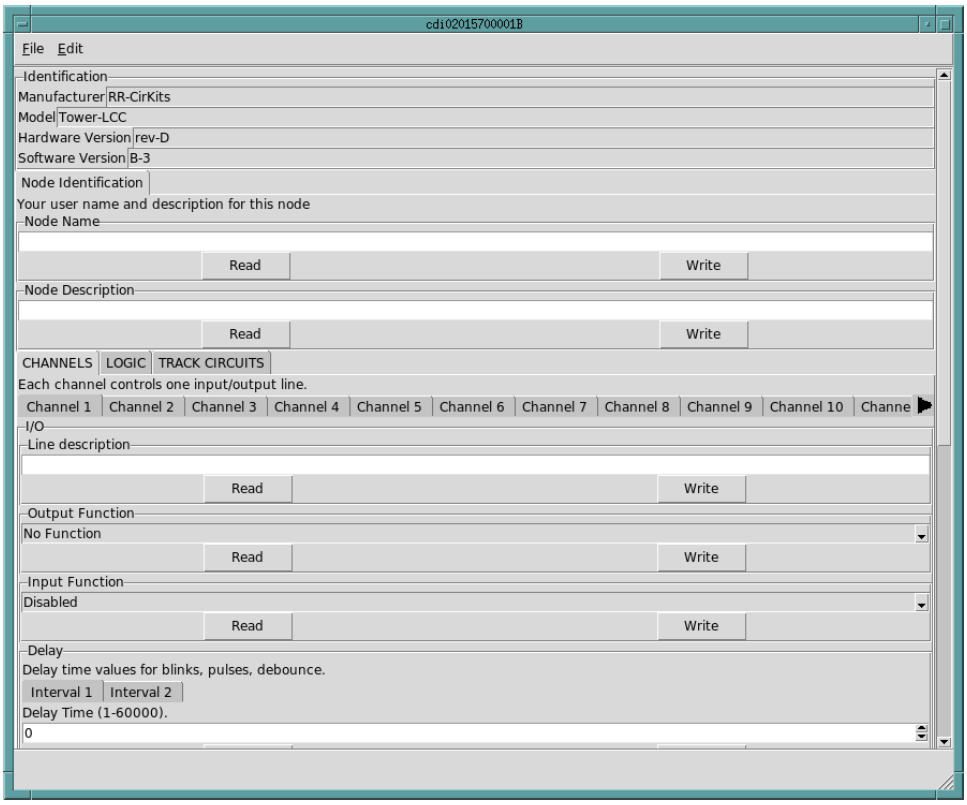


Figure 4.7 A CDI-based configuration screen for a RR-Cirkits Tower-LLC node

The generated configuration editor is a generated configuration tool customized for the selected node. All of the configuration memory locations are labeled and organized for easy access.

4.2.2 Event Tools

In addition to configuring memory, the OpenLCB can be used to manually generate event reports and to monitor the network for event production. Actuators and sensors can be tested for proper operation.

4.2.2.1 Send Event Tool

Under the `File` menu there is a **Send Event** menu item. This menu item pops up the send event dialog box, which looks like:

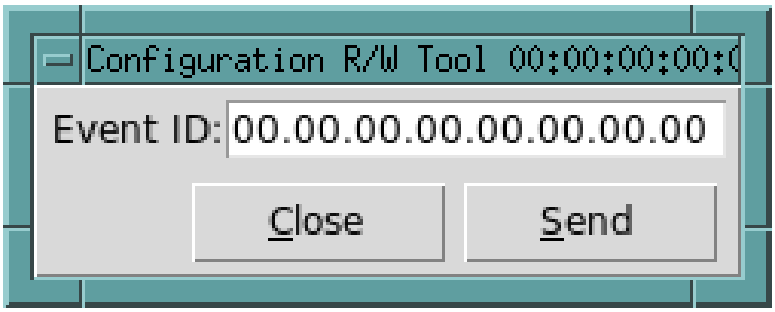


Figure 4.8 Send Event Dialog Box

This dialog box can be used to send events manually to test node consumption of the sent events.

4.2.2.2 Received Events

Additionally, if a node on the network generates an event, the OpenLCB program will display the event in a dialog box like this:

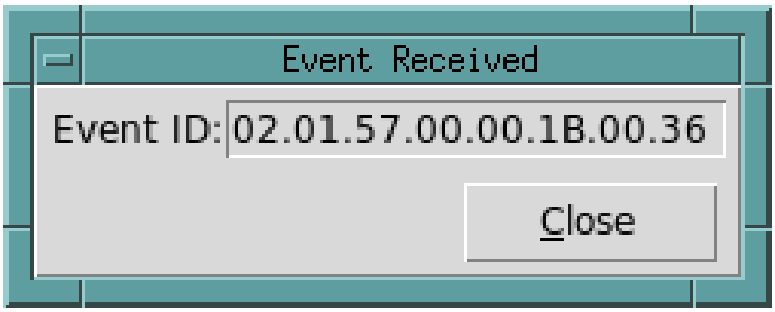


Figure 4.9 Event Received popup dialog box

Chapter 5

OpenLCB Daemons (Hubs and Virtual nodes)

A number of OpenLCB daemons are provided by the Model Railroad System. These daemons provide operational OpenLCB functionallity, including providing hubs and gateways for both real physical nodes and virtual nodes, along with several virtual nodes.

5.1 Hub Daemons

The Hub Daemons [1] create a virtual "wire" that connects multiple virtual nodes. Each node is a separately running process that has connected to the daemons network port. The hub deamon reads LCC messages from each of its connections and then writes those messages out to one (if it is specificly addressed) or all (if it is a broadcast message) of its connections. It does not write the message back out to the connection the message came from. It maintains a routing table that maps source addresses (or aliases) with source connections. Hub deamons are configured from their command line. Mostly this is the address to bind the port to and the port to bind (listen on). By default the hub deamons bind only to localhost, the loopback network device. This means that only virtual nodes running on the local machine can connect and the resultant network is "private" and local to the local machine. Optionally, the bind host (-host) can be set to 0.0.0.0. This causes the daemon to bind to all available network interfaces and make itself generally available to the whole network. [2]

There are two hub daemons that implement a OpenLCB network over Tcp/Ip and connect CAN busses connected to different host computers connected via Tcp/Ip over Ethernet. These daemons are:

- OpenLCB Tcp/Ip Hub Server The OpenLCBTcpHub daemon implememts the binary OpenLCB messaging protocol over Tcp/Ip.
- OpenLCB GridConnect Tcp/Ip Hub Server The OpenLCBGCTcpHub daemon implememts the OpenLCB messaging using the GridConnect protocol over both Tcp/Ip and using the CAN Bus over a USB/Serial connection.

Both hub daemons implement a OpenLCB network over Tcp/Ip, although using different message formats. Both also take a common set of command line arguments. The common command line arguments define the host ports and devices to bind sockets to. The GridConnect hub can also connect to both physical CAN busses (over [USB] serial ports) and other OpenLCB network hubs over Tcp/Ip. The daemons run non-interactively and log their activity to a log file.

[1] In UNIX usage, a daemon is a non-interactive process running in the background, usually (but not always) presenting some sort of connection API (like a network socket) for other processes to connect to as a way of aquiring some sort of service.

[2] If the machine has a network interface that is "public facing", this would make the daemon available on the public Internet. You should be careful, since the LCC system provides no partitular security features.

5.2 Virtual Nodes

There are several virtual nodes that implement OpenLCB nodes to provide useful functions. These daemons are:

- EventExchange node for Azatrax MRD2 boards. The OpenLCB_MRD2 daemon implememts an OpenLCB node that implements the EventExchange protocol for Azatrax MRD2 boards.
- EventExchange node for Raspberry Pi GPIO pins. The OpenLCB_PiGPIO daemon implememts an OpenLCB node that implements the EventExchange protocol for Raspberry Pi GPIO pins.
- EventExchange node for MCP23008 GPIO pins. The OpenLCB_PiMCP23008 daemon implememts an OpenLCB node that implements the EventExchange protocol for the GPIO pins on a MCP23008 I2C port expander connected to a Raspberry Pi.
- EventExchange node for MCP23017 GPIO pins. The OpenLCB_PiMCP23017 daemon implememts an OpenLCB node that implements the EventExchange protocol for the GPIO pins on a MCP23017 I2C port expander connected to a Raspberry Pi.
- EventExchange node for MCP23017 as signal heads. The OpenLCB_PiMCP23017_signal daemon implememts an OpenLCB node that implements the EventExchange protocol for the GPIO pins on a MCP23017 I2C port expander connected to a Raspberry Pi. This version groups the pins into signal heads and all pins are set to to output mode.
- EventExchange node for the quad signal head HAT. The OpenLCB_QuadSignal daemon implememts an OpenLCB node that implements the EventExchange protocol for the MCP23017-based quad signal head HAT for the Raspberry Pi. Each signal mast can have 1, 2, or 3 "heads". Each head has four "lamps" (unused lamps can be set to "None"). For a given aspect, a lamp can be on, off, blink, or reverse blink.
- EventExchange node for a SPI connected MAX7221 Signal Driver. The OpenLCB_PiSPIMax7221 daemon implememts an OpenLCB node that implements the EventExchange protocol for a SPI connected MAX7221 Signal Driver board connected to a Raspberry Pi.
- EventExchange node for virtual track circuits. The OpenLCB_TrackCircuits daemon implememts an OpenLCB node that implements virtual track circuit messaging logic using OpenLCB Events.
- EventExchange node for logic blocks. The OpenLCB_Login daemon implememts an OpenLCB node that implements logic blocks using OpenLCB Events.
- EventExchange node for a CTI Acela network. The OpenLCB_Acela daemon implememts an OpenLCB node that implements the EventExchange protocol for a CTIAcela network.
- EventExchange node for a C/MRI network. The OpenLCB_CMRI daemon implememts the EventExchange protocol for a C/MRI network.

All of these programs normally run as non-interactive daemon processes and use a configuration file in XML format to define the detailed operation of the programs. This configuration file can either be hand edited or can be edited by the programs themselves using the specific GUI configuration editor built-in to each program.

Additionally, the Dispatcher program can generate Event Exchange based CTC panel programs that connects to a OpenLCB network as nodes and produces events in response to control elements and consumes events to update track work state and control element indicators.

Not only can these nodes interact with devices on a physical OpenLCB network (such as a CAN bus), but also with each other over a virtual OpenLCB network or even both at the same time.

5.2.1 Common Node Configuration

All of the Virtual Nodes have these common configuration fields:

- An identification section, containing fields for the user supplied name and description for the node. These are free form text fields and can contain a name and description of the node.
- A transport section, containing fields for a transport constructor and options for the transport constructor. There are presently three transports. There is a **Select** button next to the constructor field to select the transport to use. The options for the transport constructor can be selected with the **Select** button next to the transport options field. The four transports are:
  - CANGridConnectOverUSBSerial: Grid Connect CAN over USB Serial
  - CANGridConnectOverTcp: Grid Connect CAN over Tcp
  - CANGridConnectOverCANSocket: Grid Connect CAN over CAN Socket
  - OpenLCBOverTcp: OpenLCB over Tcp (binary)

5.2.2 EventExchange node for Azatrax MRD2 boards.

The OpenLCB_MRD2 daemon is used to tie one or more USB connected Azatrax MRD2 boards to an OpenLCB network, tying event production to the Sense and Latch inputs of each defined connected device and, for relay equiped boards, event consumption to the Channel 1 and Channel 2 outputs of each defined connected device.

In addition to the Common Node Configuration fields the OpenLCB_MRD2 daemon has a field for a polling interval in miliseconds, defaulting to 500. This is the interval between polls of the MRD2 devices. Then for each device there is a tab containing these fields:

- description A textual description of the device. This could be the name of the block it senses.
- serial number The serial number of the device. This is printed on a sticker attached to the device.
- sense 1 on The event to send when sense 1 is activated.
- sense 1 off The event to send when sense 1 is deactivated.
- sense 2 on The event to send when sense 2 is activated.
- sense 2 off The event to send when sense 2 is deactivated.
- latch 1 on The event to send when latch 1 is activated.
- latch 1 off The event to send when latch 1 is deactivated.
- latch 2 on The event to send when latch 2 is activated.
- latch 2 off The event to send when latch 2 is deactivated.
- set chan 1 The event that triggers setting channel 1.
- set chan 2 The event that triggers setting channel 2.

5.2.2.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
<!-- XML Schema for OpenLCB_MRD2 configuration files -->
<xs:schema version="OpenLCB_MRD2 1.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_MRD2" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_MRD2 daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
              This is the node identification section.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" minOccurs="0" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="pollinterval" minOccurs="0" maxOccurs="1" />
        <xs:element name="name" minOccurs="0" maxOccurs="1" />
        <xs:element name="description" minOccurs="0" maxOccurs="1" />
        <xs:element name="device" minOccurs="0" maxOccurs="unbounded" >
          <xs:annotation>
            <xs:documentation>
              This defines one device.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="serial" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="sense1on" minOccurs="0" maxOccurs="1" />
              <xs:element name="sense1off" minOccurs="0" maxOccurs="1" />
              <xs:element name="sense2on" minOccurs="0" maxOccurs="1" />
              <xs:element name="sense2off" minOccurs="0" maxOccurs="1" />
              <xs:element name="latch1on" minOccurs="0" maxOccurs="1" />
              <xs:element name="latch1off" minOccurs="0" maxOccurs="1" />
              <xs:element name="latch2on" minOccurs="0" maxOccurs="1" />
              <xs:element name="latch2off" minOccurs="0" maxOccurs="1" />
              <xs:element name="setchan1" minOccurs="0" maxOccurs="1" />
              <xs:element name="setchan2" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.2.3 EventExchange node for Raspberry Pi GPIO pins.

The OpenLCB_PiGPIO daemon is used to tie one or more of a Raspberry Pi's GPIO pins to event production (input pins) or event consumption (output pins).

In addition to the Common Node Configuration fields the OpenLCB_PiGPIO daemon has a field for a polling interval in miliseconds, defaulting to 500. This is the interval between polls of the GPIO Pins. Then for each pin there is a tab containing these fields:

- description A textual description of the pin.
- number The number of the pin.
- mode The mode of the pin, one of disabled, in, out, high, low.
- pin in 0 The event to send when the pin goes to 0.
- pin in 1 The event to send when the pin goes to 1.
- pin out 0 The event to set the pin to 0.
- pin out 1 The event to set the pin to 1.

5.2.3.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
<!-- XML Schema for OpenLCB_PiGPIO configuration files -->
<xs:schema version="OpenLCB_PiGPIO 1.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_PiGPIO" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_PiGPIO daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
          <xs:annotation>
```

```
            <xs:documentation>
              This is the node identification section.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" minOccurs="0" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="pollinterval" minOccurs="0" maxOccurs="1" />
        <xs:element name="pin" minOccurs="0" maxOccurs="unbounded" >
          <xs:annotation>
            <xs:documentation>
              This defines one pin.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="number" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="mode" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinin0" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinin1" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinout0" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinout1" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.2.4 EventExchange node for MCP23008 GPIO pins.

The OpenLCB_PiMCP23008 daemon is used to tie one or more of a MCP23008's GPIO pins to event production (input pins) or event consumption (output pins). A MCP23008 is a 8 bit I2C port expander that can be connected to a Raspberry Pi.

In addition to the Common Node Configuration fields the OpenLCB_PiMCP23008 daemon has a field for a polling interval in miliseconds, defaulting to 500. This is the interval between polls of the GPIO Pins. There is also a field containing the low 3 bits of the address of the MCP23008's I2C address (the default is 7). Then for each pin there is a tab containing these fields:

- description A textual description of the pin.
- number The number of the pin.
- mode The mode of the pin, one of disabled, in, out, high, low.
- pin in 0 The event to send when the pin goes to 0.
- pin in 1 The event to send when the pin goes to 1.
- pin out 0 The event to set the pin to 0.
- pin out 1 The event to set the pin to 1.

5.2.4.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
<!-- XML Schema for OpenLCB_PiMCP23008 configuration files -->
<xs:schema version="OpenLCB_PiMCP23008 1.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_PiMCP23008" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_PiMCP23008 daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
              This is the node identification section.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" minOccurs="0" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="pollinterval" minOccurs="0" maxOccurs="1" />
        <xs:element name="i2caddress" minOccurs="0" maxOccurs="1" />
        <xs:element name="pin" minOccurs="0" maxOccurs="unbounded" >
          <xs:annotation>
            <xs:documentation>
              This defines one pin.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="number" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="mode" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinin0" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinin1" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinout0" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinout1" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.2.5 EventExchange node for MCP23017 GPIO pins.

The OpenLCB_PiMCP23017 daemon is used to tie one or more of a MCP23017's GPIO pins to event production (input pins) or event consumption (output pins). A MCP23017 is a 16 bit I2C port expander that can be connected to a Raspberry Pi.

In addition to the Common Node Configuration fields the OpenLCB_PiMCP23017 daemon has a field for a polling interval in miliseconds, defaulting to 500. This is the interval between polls of the GPIO Pins. There is also a field containing the low 3 bits of the address of the MCP23017's I2C address (the default is 7). Then for each pin there is a tab containing these fields:

- description A textual description of the pin.
- number The number of the pin.
- mode The mode of the pin, one of disabled, in, out, high, low.
- pin in 0 The event to send when the pin goes to 0.
- pin in 1 The event to send when the pin goes to 1.
- pin out 0 The event to set the pin to 0.
- pin out 1 The event to set the pin to 1.

5.2.5.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
<!-- XML Schema for OpenLCB_PiMCP23017 configuration files -->
<xs:schema version="OpenLCB_PiMCP23017 1.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_PiMCP23017" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_PiMCP23017 daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
              This is the node identification section.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
```

```
            <xs:sequence>
              <xs:element name="name" minOccurs="0" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="pollinterval" minOccurs="0" maxOccurs="1" />
        <xs:element name="i2caddress" minOccurs="0" maxOccurs="1" />
        <xs:element name="pin" minOccurs="0" maxOccurs="unbounded" >
          <xs:annotation>
            <xs:documentation>
              This defines one pin.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="number" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="mode" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinin0" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinin1" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinout0" minOccurs="0" maxOccurs="1" />
              <xs:element name="pinout1" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.2.6 EventExchange node for MCP23017 as signal heads.

The OpenLCB_PiMCP23017_signal daemon is used to tie groups of a MCP23017's GPIO pins into signal heads and all pins are set to to output mode. A MCP23017 is a 16 bit I2C port expander that can be connected to a Raspberry Pi.

In addition to the Common Node Configuration fields the OpenLCB_PiMCP23017 daemon has a field containing the low 3 bits of the address of the MCP23017's I2C address (the default is 7). Then for each signal there is a tab containing these fields:

- description A textual description of the signal.
- number The number of the first pin used by the signal.
- ledcount The number of LEDs used by the signal.
- common Either anode or cathode to indicate if the LEDs are wired as common anode or common cathode.
- zero or more Aspect tabs, containing:
  - eventid The event ID for this aspect.
  - bits The aspect's bit field as a binary number (letter B followed by 1s (on) and 0s (off).

5.2.6.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
<!-- XML Schema for OpenLCB_PiMCP23017_signal configuration files -->
<xs:schema version="OpenLCB_PiMCP23017_signal 1.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_PiMCP23017_signal" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_PiMCP23017_signal daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
              This is the node identification section.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" minOccurs="0" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="i2caddress" minOccurs="0" maxOccurs="1" />
        <xs:element name="signal" minOccurs="0" maxOccurs="unbound">
          <xs:annotation>
            <xs:documentation>
              This defines one signal.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="number" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="ledcount" minOccurs="1" maxOccurs="1" />
              <xs:element name="common" minOccurs="1" maxOccurs="1" />
              <xs:element name="aspect" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="eventid" minOccurs="1" maxOccurs="1" />
                    <xs:element name="bits" minOccurs="1" maxOccurs="1" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
```

```
  </xs:element>
</xs:schema>
```

5.2.7 EventExchange node for the quad signal head HAT.

The OpenLCB_QuadSignal daemon implememts an OpenLCB node that implements the EventExchange protocol for the MCP23017-based quad signal head HAT for the Raspberry Pi. Each signal mast can have 1, 2, or 3 "heads". Each head has four "lamps" (unused lamps can be set to "None"). For a given aspect, a lamp can be on, off, blink, or reverse blink. In addition to the Common Node Configuration fields the OpenLCB_PiMCP23017 daemon has a field containing the low 3 bits of the address of the MCP23017's I2C address (the default is 7). Then for each signal mast there is a tab containing these fields:

- description A textual description of the signal.
- zero or more Aspect tabs, each containing:
  - eventid The event ID for this aspect.
  - name The name of the aspect.
  - one or more Head tabs, each containing:
    - four Lamp tabs, each containing:
      - id The lamp id, one of None, H1-G, H1-Y, H1-R, H1-L, H2-G, H2-Y, H2-R, H2-L, H3-G, H3-Y, H3-R, H3-L, H4-G, H4-Y, H4-R, or H4-L.
      - effect The lamp effect, one of off, on, blink, or reverseblink.

5.2.7.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
<!-- XML Schema for OpenLCB_PiMCP23017_signal configuration files -->
<xs:schema version="OpenLCB_PiMCP23017_signal 1.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_PiMCP23017_signal" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_PiMCP23017_signal daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
```

```
            This is the node identification section.
          </xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" minOccurs="0" maxOccurs="1" />
            <xs:element name="description" minOccurs="0" maxOccurs="1" />
          </xs:sequence>
        </xs:complexType></xs:complexType>
      </xs:element>
      <xs:element name="i2caddress" minOccurs="0" maxOccurs="1" />
      <xs:element name="mast" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="description" minOccurs="0" maxOccurs="1" />
            <xs:element name="aspect" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="eventid" minOccurs="1" maxOccurs="1" />
                  <xs:element name="name" minOccurs="1" maxOccurs="1" />
                  <xs:element name="head" minOccurs="1" maxOccurs="unbounded">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="lamp" minOccurs="4" maxOccurs="4" >
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="id" minOccurs="1" maxOccurs="1" />
                              <xs:element name="effect" minOccurs="1" maxOccurs="1" />
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

5.2.8 EventExchange node for a SPI connected MAX7221 Signal Driver.

The OpenLCB_PiSPIMax7221 daemon is used to implement upto 8 signal masts using a SPI connected MAX7221 LED driver. Each `digit` of the MAX7221 corresponds to one signal mast and each segment of each digit corresponds to one LED.

In addition to the Common Node Configuration fields the OpenLCB_PiSPIMax7221 daemon has a field to select the SPI channel (0 or 1) and tabs for each signal (up to 8). Each signal has a signal number (1 through 8), a textual description, and any number (at least one) of aspects. Each aspect contain an event id and an eight bit field defining the on (1) or off (0) LEDS, typically top to bottom.

5.2.8.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
```

```
<!-- XML Schema for OpenLCB_PiSPIMax7221 configuration files -->
<xs:schema version="OpenLCB_PiSPIMax7221 1.0"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_PiSPIMax7221" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_PiSPIMax7221
        daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
              This defines the transport to use for this node.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
           <xs:annotation>
             <xs:documentation>
               This is the node identification section.
             </xs:documentation>
           </xs:annotation>
           <xs:complexType>
             <xs:sequence>
               <xs:element name="name" minOccurs="0" maxOccurs="1" />
               <xs:element name="description" minOccurs="0" maxOccurs="1" />
             </xs:sequence>
           </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="spichannel" minOccurs="0" maxOccurs="1" />
        <xs:element name="signal" minOccurs="1" maxOccurs="8" >
          <xs:annotation>
            This defines one signal.
          </xs:documentation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="number" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="aspect" minOccurs="1" maxOccurs="unbounded" >
                 <xs:complexType>
                   <xs:element name="eventid" minOccurs="1" maxOccurs="1" />
                   <xs:element name="bits" minOccurs="1" maxOccurs="1" />
                 </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.2.9 EventExchange node for virtual track circuits.

The OpenLCB_TrackCircuits daemon is used to implement one or more virtual track circuits. Each track circuit can emit a code event in response to an event and can emit an event in response to a code event, possibly prefixed with a Code

1 Start event.

In addition to the Common Node Configuration fields the OpenLCB_TrackCircuits daemon has tabs for each track, containing these fields:

- Description A textual description of the track
- Track Service Enabled or Disabled
- Command tabs Zero or more command tabs which map a received event to a track code.
- Transmit Group Base Event The track code will be added to this event.
- Receive Group Base Event This is the base track code reciever event.
- Code 1 Start Event The event to send when a Code 1 Start occur.
- Action tabs Zero or more action tabs which map an event to send when a track code received.

The track codes defined for transmitters and receivers are:

- `None` No track code.
- `Code7` Clear
- `Code4` Advance Approach
- `Code3` Approach Limited
- `Code8` Approach Medium
- `Code2` Approach
- `Code9` Approach Slow
- `Code6` Accelerated Tumble Down
- `Code5_occupied` Non-Vital (occupied)
- `Code5_normal` Non-Vital (normal)
- `CodeM_failed` Power/Lamp (failed)
- `CodeM_normal` Power/Lamp (normal)

5.2.9.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
 <!-- XML Schema for OpenLCB_TrackCircuits configuration files -->
 <xs:schema version="OpenLCB_TrackCircuits 1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <xs:element name="OpenLCB_TrackCircuits" minOccurs="1" maxOccurs="1">
     <xs:annotation>
       <xs:documentation>
         This is the configuration container for the OpenLCB_TrackCircuits
         daemon.
       </xs:documentation>
```

```
</xs:annotation>
<xs:complexType>
  <xs:sequence>
    <xs:element name="transport" minOccurs="1" maxOccurs="1">
      <xs:annotation>
        <xs:documentation>
          This defines the transport to use for this node.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
          <xs:element name="options" minOccurs="1" maxOccurs="1" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="identification" minOccurs="0" maxOccurs="1">
       <xs:annotation>
         <xs:documentation>
           This is the node identification section.
         </xs:documentation>
       </xs:annotation>
       <xs:complexType>
         <xs:sequence>
           <xs:element name="name" minOccurs="0" maxOccurs="1" />
           <xs:element name="description" minOccurs="0" maxOccurs="1" />
         </xs:sequence>
       </xs:complexType>
    </xs:element>
    <xs:element name="track" minOccurs="0" maxOccurs="unbounded" >
      <xs:annotation>
        <xs:documentation>
          This defines one track.
        </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="description" minOccurs="0" maxOccurs="1" />
          <xs:element name="enabled" minOccurs="0" maxOccurs="1" />
          <xs:element name="transmitter" minOccurs="0"
                      maxOccurs="unbounded" >
            <xs:complexType>
              <xs:sequence>
                <xs:element name="code" minOccurs="1" maxOccurs="1" />
                <xs:element name="eventid" minOccurs="1"
                            maxOccurs="1" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="transmitbaseevent" minOccurs="0"
                      maxOccurs="1" />
          <xs:element name="receivebaseevent" minOccurs="0"
                      maxOccurs="1" />
          <xs:element name="code1startevent" minOccurs="0"
                      maxOccurs="1" />
          <xs:element name="receiver" minOccurs="0"
                      maxOccurs="unbounded" >
            <xs:complexType>
              <xs:sequence>
                <xs:element name="code" minOccurs="1" maxOccurs="1" />
                <xs:element name="eventid" minOccurs="1"
                            maxOccurs="1" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:schema>
```

5.2.10 EventExchange node for logic blocks.

The OpenLCB_Logic daemon is used to implement one or more logic blocks. Each logic can be standalone or part of a mast or ladder group.

In addition to the Common Node Configuration fields the OpenLCB_Logic daemon has tabs for each logic block, containing these fields:

- Description A textual description of the track
- The Group Type, one of `single` (Single or last), `mast` (Mast Group), or `ladder` (Ladder Group).
- An event to set variable 1 true.
- An event to set variable 1 false.
- The logic function, one of `and` (V1 and V2), `or` (V1 or V2), `xor` (V1 xor V2), `andch` (V1 and V2 change), `orch` (V1 or V2 change), `then` (V1 then V2), or `true`.
- An event to set variable 2 true.
- An event to set variable 2 false.
- The delay in miliseconds (0 means no delay).
- Whether the delay is retriggerable.
- Four (4) action tabs, each with a delay flag and an event.

5.2.10.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
 <!-- XML Schema for OpenLCB_Logic configuration files -->
 <xs:schema version="OpenLCB_Logic 1.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <xs:element name="OpenLCB_Logic" minOccurs="1" maxOccurs="1">
     <xs:annotation>
       <xs:documentation>
         This is the configuration container for the OpenLCB_Logic
         daemon.
       </xs:documentation>
     </xs:annotation>
     <xs:complexType>
       <xs:sequence>
         <xs:element name="transport" minOccurs="1" maxOccurs="1">
           <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
           </xs:annotation>
```

```
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
           <xs:annotation>
             <xs:documentation>
               This is the node identification section.
             </xs:documentation>
           </xs:annotation>
           <xs:complexType>
             <xs:sequence>
               <xs:element name="name" minOccurs="0" maxOccurs="1" />
               <xs:element name="description" minOccurs="0" maxOccurs="1" />
             </xs:sequence>
           </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="logic" minOccurs="0" maxOccurs="unbounded" >
          <xs:annotation>
            <xs:documentation>
              This defines one logic block.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="grouptype" minOccurs="1" maxOccurs="1" />
              <xs:element name="v1onevent" minOccurs="0" maxOccurs="1" />
              <xs:element name="v1offevent" minOccurs="0" maxOccurs="1" />
              <xs:element name="logicfunction" minOccurs="1" maxOccurs="1" />
              <xs:element name="v2onevent" minOccurs="0" maxOccurs="1" />
              <xs:element name="v2offevent" minOccurs="0" maxOccurs="1" />
              <xs:element name="delay" minOccurs="0" maxOccurs="1" />
              <xs:element name="retriggerable" minOccurs="0" maxOccurs="1" />
              <xs:element name="action1delay" minOccurs="0" maxOccurs="1" />
              <xs:element name="action1event" minOccurs="0" maxOccurs="1" />
              <xs:element name="action2delay" minOccurs="0" maxOccurs="1" />
              <xs:element name="action2event" minOccurs="0" maxOccurs="1" />
              <xs:element name="action3delay" minOccurs="0" maxOccurs="1" />
              <xs:element name="action3event" minOccurs="0" maxOccurs="1" />
              <xs:element name="action4delay" minOccurs="0" maxOccurs="1" />
              <xs:element name="action4event" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.2.11 EventExchange node for a CTI Acela network.

The OpenLCB_Acela daemon is used to tie a CTI Acela network to an OpenLCB network, tying event production to the inputs (sensors) and outputs (controls and signals) connected to a CTI Acela network.

In addition to the Common Node Configuration fields the OpenLCB_Acels daemon has tabs for each Control, Signal, or Sensor. Each type has a numerical address and a textual description.

You will want to read the "The Acela Network Bridge Programmer's Guide" for an explaination of some of the terminolgy used here.

In addition each Control has these fields:

- Pulse Width in 10ths of a second. Used with the Pulse on and Pulse off events.
- Blink Period in 10ths of a second. Used with the Blink and Reverse Blink events.
- Activate eventid
- Deactivate eventid
- Pulse on eventid
- Pulse off eventid
- Blink eventid
- Reverse Blink eventid

In addition each Signal has these fields:

- Signal command, one of Signal2, Signal3, or Signal4. Signal2 uses two consequential outputs and assumes a bi-color led (red/green) and simulates yellow Signal3 uses three consequential outputs and assumes three descrete lamps or leds. Signal4 uses four consequential outputs and assumes four descrete lamps or leds.
- Plus zero or more Aspect tabs. Each Aspect tab has a event field and an argument list for a signal. This 2 or 3 elements for Signal2, three elements for Signal3 and four elements for Signal4. Eacl element defines one lamp or led and is one of: on, off, blink, revblink.

There are also three common fields for all signals:

- Signal blink rate in 10ths of a second.
- Yellow Hue
- Signal brightless

In addition each sensor has these fields:

- Filter Threshold
- Filter Select
- Polarity
- The on eventid
- The off eventid

5.2.11.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
<!-- XML Schema for OpenLCB_Acela configuration files -->
<xs:schema version="OpenLCB_Acela 1.0"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_Acela" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_Acela daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
              This is the node identification section.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" minOccurs="0" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="acelaport" minOccurs="1" maxOccurs="1" />
        <xs:element name="blinkrate" minOccurs="0" maxOccurs="1" />
        <xs:element name="yellowhue" minOccurs="0" maxOccurs="1" />
        <xs:element name="brightness" minOccurs="0" maxOccurs="1" />
        <xs:element name="control" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>
              This defines one Control.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="address" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="pulsewidth" minOccurs="0" maxOccurs="1" />
              <xs:element name="blinkperiod" minOccurs="0" maxOccurs="1" />
              <xs:element name="activate" minOccurs="0" maxOccurs="1" />
              <xs:element name="deactivate " minOccurs="0" maxOccurs="1" />
              <xs:element name="pulseon " minOccurs="0" maxOccurs="1" />
              <xs:element name="pulseoff " minOccurs="0" maxOccurs="1" />
              <xs:element name="blink " minOccurs="0" maxOccurs="1" />
              <xs:element name="revblink" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="signal" minOccurs="0" maxOccurs="unbounded">
```

```
          <xs:annotation>
            <xs:documentation>
              This defines one Signal.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="address" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="pulsewidth" minOccurs="0" maxOccurs="1" />
              <xs:element name="aspect" minOccurs="0"
                          maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="eventid" minOccurs="0"
                                maxOccurs="1" />
                    <xs:element name="arglist" minOccurs="0"
                                maxOccurs="1" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="sensor" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
            <xs:documentation>
              This defines one Sensor.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="address" minOccurs="1" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
              <xs:element name="filterthresh" minOccurs="0"
                                              maxOccurs="1" />
              <xs:element name="filterselect" minOccurs="0"
                                              maxOccurs="1" />
              <xs:element name="polarity" minOccurs="0" maxOccurs="1" />
              <xs:element name="onevent" minOccurs="0" maxOccurs="1" />
              <xs:element name="offevent" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

5.2.12 EventExchange node for a C/MRI network.

The OpenLCB_CMRI daemon implememts the EventExchange protocol for a C/MRI network, tying event production to the inputs and outputs connected to a C/MRI network.

In addition to the Common Node Configuration fields the OpenLCB_CMRI has these global fields:

- port The serial port the C/MRI network is on.
- baud The baud rate to use.
- maxtries The maximum number of retries. Then there are zero or more node tabs with these fields:
- description The description of the node.

- address This is the address of the node.
- type The type of card (SUSIC, USIC, or SMINI).
- cardmap The card map list (SUSIC or USIC).
- yellowmap The yellow map list (SMINI).
- numberofyellow The number of yellows (SMINI).
- inputports The number of 8-bit input ports.
- outputports The number of 8-bit output ports.
- delay The delay value to use (older SUSIC and USIC nodes).
- zero or more input tabs:
  - eventid The event to produce.
  - byte The byte (port) offset.
  - mask The mask value.
  - comp The comparison operator (== or !=).
  - value The value to compare to.
- zero or more output tabs:
  - eventid The event to consume.
  - byte The byte (port) offset.
  - mask The mask value.
  - value The value to write.

5.2.12.1 XML Schema for configuration files

```
<?xml version="1.0" ?>
<?xml-stylesheet href="schema2xhtml.xsl" type="text/xsl" ?>
<!-- XML Schema for OpenLCB_Acela configuration files -->
<xs:schema version="OpenLCB_Acela 1.0"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="OpenLCB_Acela" minOccurs="1" maxOccurs="1">
    <xs:annotation>
      <xs:documentation>
        This is the configuration container for the OpenLCB_Acela daemon.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="transport" minOccurs="1" maxOccurs="1">
          <xs:annotation>
             <xs:documentation>
               This defines the transport to use for this node.
             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="constructor" minOccurs="1" maxOccurs="1" />
              <xs:element name="options" minOccurs="1" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
```

```
        <xs:element name="identification" minOccurs="0" maxOccurs="1">
          <xs:annotation>
            <xs:documentation>
              This is the node identification section.
            </xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" minOccurs="0" maxOccurs="1" />
              <xs:element name="description" minOccurs="0" maxOccurs="1" />
            </xs:sequence>
          </xs:complexType></xs:complexType>
        </xs:element>
        <xs:element name="port" minOccurs="1" maxOccurs="1" />
        <xs:element name="baud" minOccurs="1" maxOccurs="1" />
        <xs:element name="maxtries" minOccurs="1" maxOccurs="1" />
        <xs:element name="node" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="description" minOccurs="1" maxOccurs="1" />
              <xs:element name="address" minOccurs="1" maxOccurs="1" />
              <xs:element name="type" minOccurs="1" maxOccurs="1" />
              <xs:element name="cardmap" minOccurs="0" maxOccurs="1" />
              <xs:element name="yellowmap" minOccurs="0" maxOccurs="1" />
              <xs:element name="numberofyellow" minOccurs="0" maxOccurs="1" />
              <xs:element name="inputports" minOccurs="1" maxOccurs="1" />
              <xs:element name="outputports" minOccurs="1" maxOccurs="1" />
              <xs:element name="delay" minOccurs="1" maxOccurs="1" />
              <xs:element name="input" minOccurs="0" maxOccurs="unbounded" >
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="eventid" minOccurs="1" maxOccurs="1" />
                    <xs:element name="byte" minOccurs="1" maxOccurs="1" />
                    <xs:element name="mask" minOccurs="1" maxOccurs="1" />
                    <xs:element name="comp" minOccurs="1" maxOccurs="1" />
                    <xs:element name="value" minOccurs="1" maxOccurs="1" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="output" minOccurs="0" maxOccurs="unbounded" >
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="eventid" minOccurs="1" maxOccurs="1" />
                    <xs:element name="byte" minOccurs="1" maxOccurs="1" />
                    <xs:element name="mask" minOccurs="1" maxOccurs="1" />
                    <xs:element name="value" minOccurs="1" maxOccurs="1" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Chapter 6

Offline LCC Node Editor Reference

This program makes use of the CDI Configuration Tool to edit LCC Node backup configuration files without being connected to a LCC network.

6.1 Command Line Parameters and Options

This program takes some optional options and at least one required parameter.

6.1.1 Options

6.1.2 X11 Resource Options

- -colormap: Colormap for main window
- -display: Display to use
- -geometry: Initial geometry for window
- -name: Name to use for application
- -sync: Use synchronous mode for display server
- -visual: Visual for main window
- -use: Id of window in which to embed application

6.1.2.1 Other options

- -help Print a short help message and exit.
- -debug Turn on debug output.

6.1.3 Parameters

There is one required parameter, the file containing the CDI XML for the nodes to be edited. Additional parameters are the config files to be edited.

6.2 Main GUI Elements

In addition to editing LCC Node backup config files, this program also can manage a Layout Control Database, just like the OpenLCB (see OpenLCB Program Reference) and Dispatcher (see Dispatcher Reference) programs. Its `File` menu contains items to load and save a layout control database, and its `Edit` menu contains items to create layout control elements. See the OpenLCB Program Reference documentation for info on these menu items. Additionally, the `Open` item on the `File` menu will open additional config files to edit.

The main GUI contains the table of Layout Control elements in the currently loaded layout control database, along with edit boxes for these elements.

Chapter 7

Layout Control Database

This database is an XML file containing a mapping of Layout Control elements and LCC Event Ids. The database contains turnouts, blocks, signals, sensors, and controls. The LayoutDB to JMRI Tables converter program can convert a Layout Control Database to a JMRI Table file.

7.1 Turnouts

The `turnout` tag describes a turnout. Child tags include:

- `name` This holds the name of the turnout.
- `motor` This holds the motor event ids (consumed by the turnout). Under the motor tag are two child tags:
  - `normal` This holds the normal event id.
  - `reverse` This holds the reverse event id.
- `points` This holds the points sense event ids (produced by the turnout). Under the points tag are two child tags:
  - `normal` This holds the normal event id.
  - `reverse` This holds the reverse event id.

7.2 Blocks

The `block` tag describes a block. Child tags include:

- `name` This holds the name of the block.
- `occupied` This holds the (produced) occupied event id,
- `clear` This holds the (produced) clear event id,

7.3 Signals

The `signal` tag describes a signal. Child tags include:

- `name` This holds the name of the signal.
- `aspect` This holds an aspect of the signal. A signal can have zero or more of these tags. Child tags include:
  - `name` This holds the name of the aspect.
  - `event` This holds the (consumed) event id to set the aspect.
  - `look` This contains the look of the aspect, typicaly a list of colors.

7.4 Sensors

The `sensor` tag describes a generic sensor. Child tags include:

- `name` This holds the name of the sensor.
- `on` This holds the (produced) event id when the sensor goes on (is activated).
- `off` This holds the (produced) event id when the sensor goes off (is
- deactivated).

7.5 Controls

The `control` tag describes a generic control. Child tags include:

- `name` This holds the name of the sensor.
- `on` This holds the (consumed) event id to turn the control on (activate).
- `off` This holds the (consumed) event id to turn the control off (deactivated).

Chapter 8

Azatrax Test Programs Reference

These programs can be used to test the various boards made by Azatrax. These include the MRD2-S and MRD2-U boards, which are infrared sensor units with USB interfaces. The MRD2-S includes relays for operating switch motors, power relays, or signals. The MRD2-U contain just a pair of detectors. Azatrax also makes the SR4 board, which is a quad set of solid state relays. Also planned are boards to control stall motor type switch machines and signal driver boards.

8.1 MRD Test Program Reference

This program is the basic test program and can be used to test basic functionality of either a MRD2-S or MRD2-U unit. There are buttons for each of the commands that can be sent, plus a display area showing the current state data for the unit.

8.1.1 Synopsis

```
MRDTest [X11 Resource Options]
```

This program takes no parameters.

8.1.2 Main GUI Screen

The MRDTest main GUI is shown here:

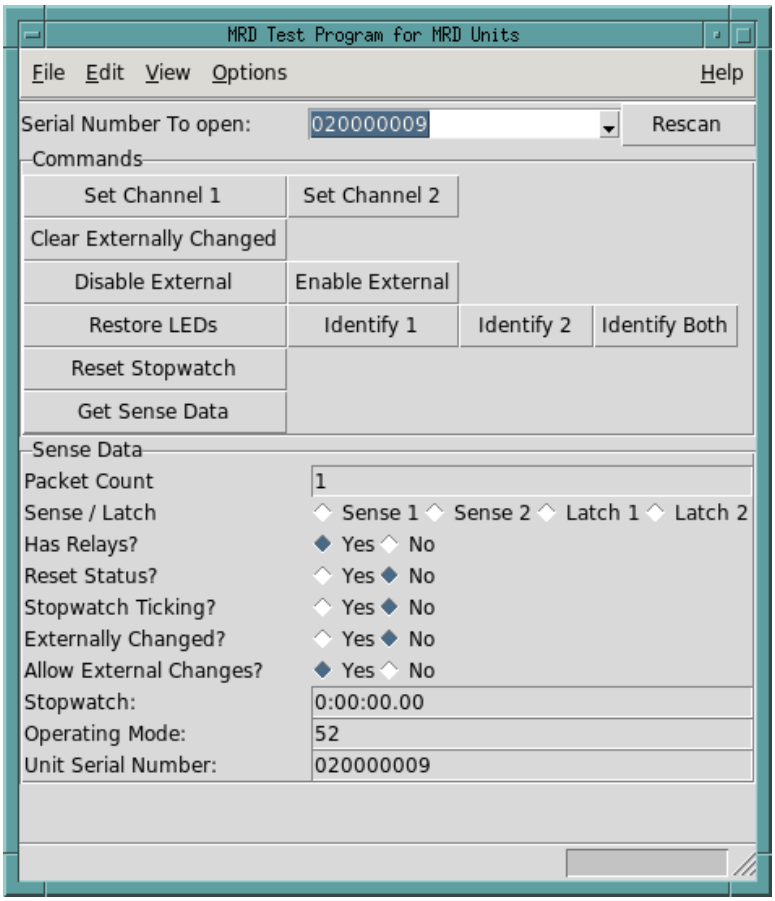


Figure 8.1 MRDTest Main GUI Screen

The upper half contains buttons to invoke each of the commands that the MRD-2 unit understands and the lower half displays the unit's sense data.

8.2 MRD Sensor Loop Reference

This program loops, reading the unit sense data at 500 millisecond intervals, displaying the state of the Sense and Latch bits, plus whether or not the stopwatch is ticking and the current stopwatch time value.

8.2.1 Synopsis

```
MRDSensorLoop [X11 Resource Options] sensorSerialNumber
```

This program takes one parameter, the serial number of the MRD2-S or MRD2-U unit to test. The program runs until exited or until the MRD2-S MRD2-U unit is unplugged.

8.2.2 Main GUI Screen

The MRDSensorLoop main GUI is shown here:

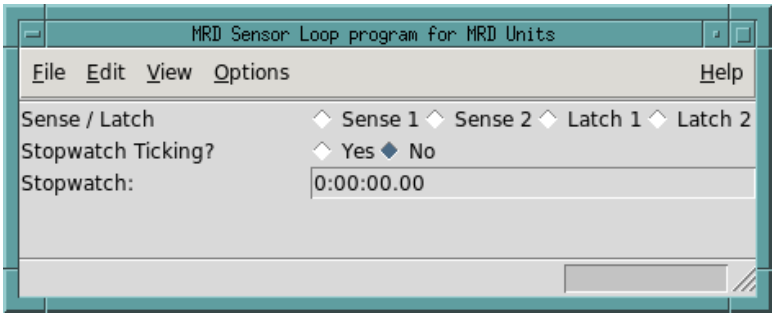


Figure 8.2 MRDSensorLoop Main GUI Screen

This screen shows the current state of the MRD2 unit. It is updated every 500 miliseconds (.5 seconds).

8.3 SR4 Test Program Reference

This program is the basic test program and can be used to test basic functionality of a SR4 unit. There are buttons for each of the commands that can be sent, plus a display area showing the current state data for the unit.

8.3.1 Synopsis

```
SR4Test [X11 Resource Options]
```

Th9iis program takes no parameters.

8.3.2 Main GUI Screen

The SR4Test main GUI is shown here:

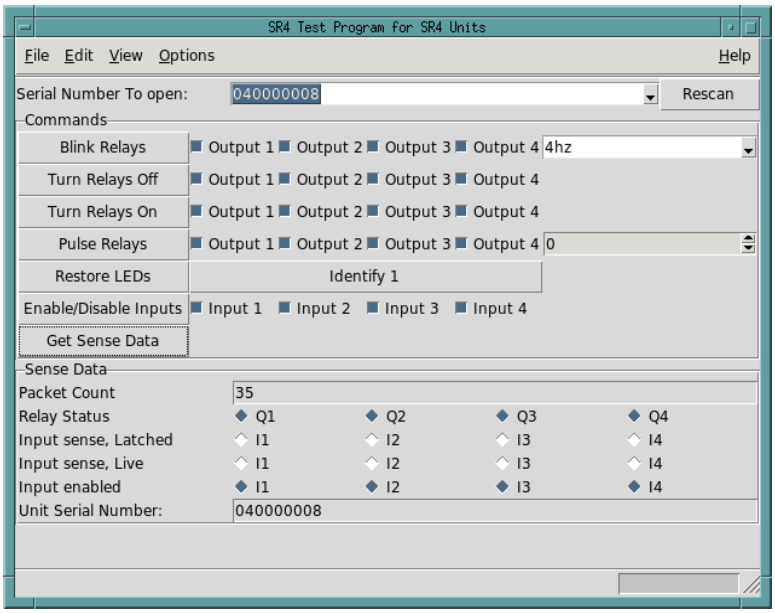


Figure 8.3 SR4Test Main GUI Screen

The upper half contains buttons to invoke each of the commands that the SR4 unit understands and the lower half displays the unit's sense data.

8.4 SL2 Test Program Reference

This program is the basic test program and can be used to test basic functionality of a SL2 unit. There are buttons for each of the commands that can be sent, plus a display area showing the current state data for the unit.

8.4.1 Synopsis

```
SL2Test [X11 Resource Options]
```

This program takes no parameters.

8.4.2 Main GUI Screen

The SL2Test main GUI is shown here:

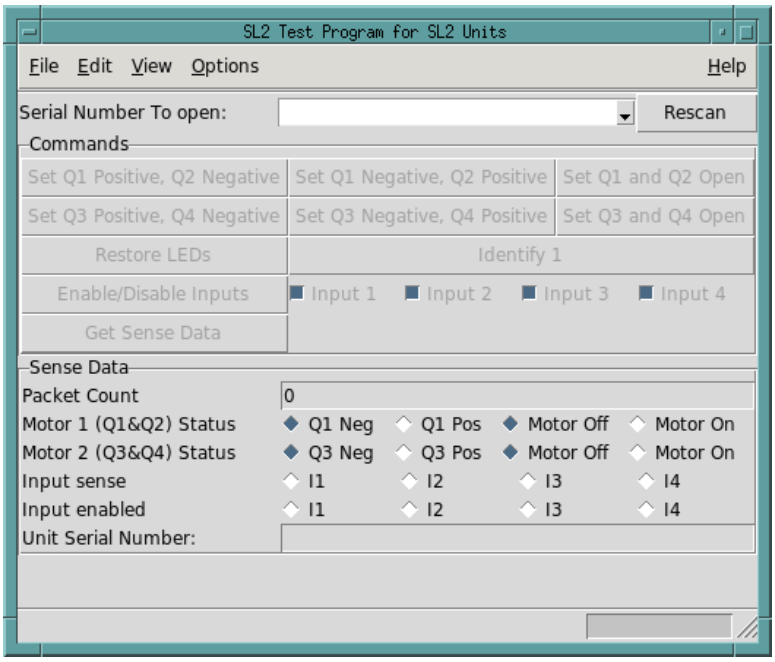


Figure 8.4 SL2Test Main GUI Screen

The upper half contains buttons to invoke each of the commands that the SL2 unit understands and the lower half displays the unit's sense data.

8.5 Azatrax Device Map Reference

This program is a GUI program for mapping Azatrax units. It creates and updates a text file that maps device serial numbers to names and descriptions. This file can be used as a reference when writing scripts and programs that use these devices.

8.5.1 Synopsis

```
AzatraxDeviceMap [X11 Resource Options] [mapfile]
```

This program takes an optional mapfile as its sole parameter

8.5.2 Main GUI Screen

The AzatraxDeviceMap main GUI is shown here:

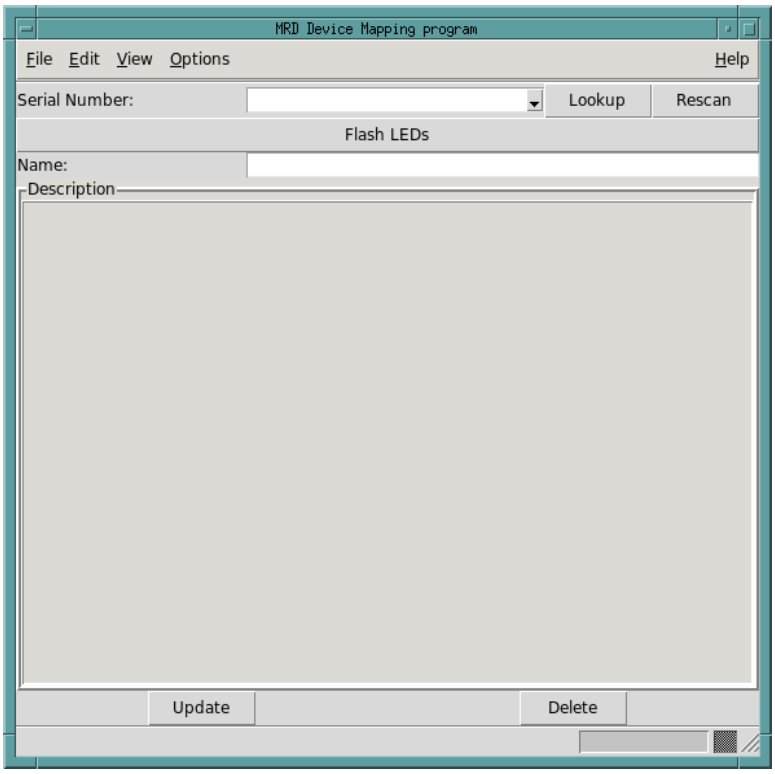


Figure 8.5 AzatraxDeviceMap Main GUI Screen

At the top is a pulldown list of discovered Azatrax unit serial numbers, which can be selected. The LEDs on the selected unit can be flashed to identify which unit it is. The unit can be given a name and a description in the fields supplied.

Chapter 9

XPressNet Throttle

The XPressNetThrottle program is a simple program that provides a "virtual" replacement for a LM50 or LM100 on your computer screen.

9.1 Main GUI

Its basic GUI in Throttle Mode is shown here:

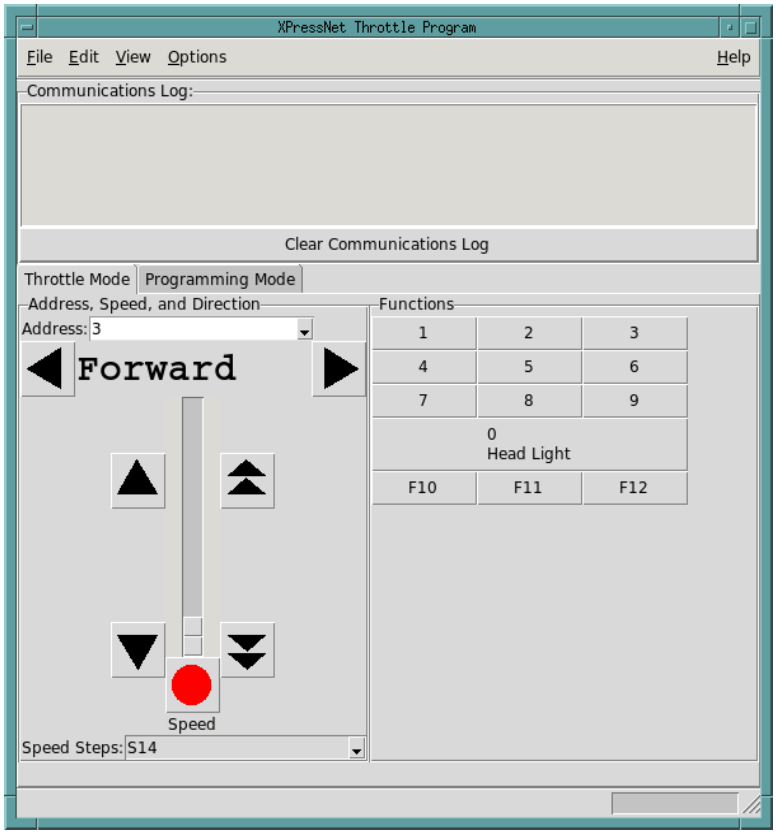


Figure 9.1 XPressNetThrottle Main GUI in Throttle Mode

On the left is a field to enter the locomotive's address, and buttons for selecting the locomotive's direction and a slider for selecting the locomotive's speed. On the right is an array of buttons to select the locomotive's function bits. By default, the locomotive address is set to 3 but you can enter a different address. The controls are pretty self explainitory.

9.2 Programming Mode

In programming mode, the Main GUI looks like this:

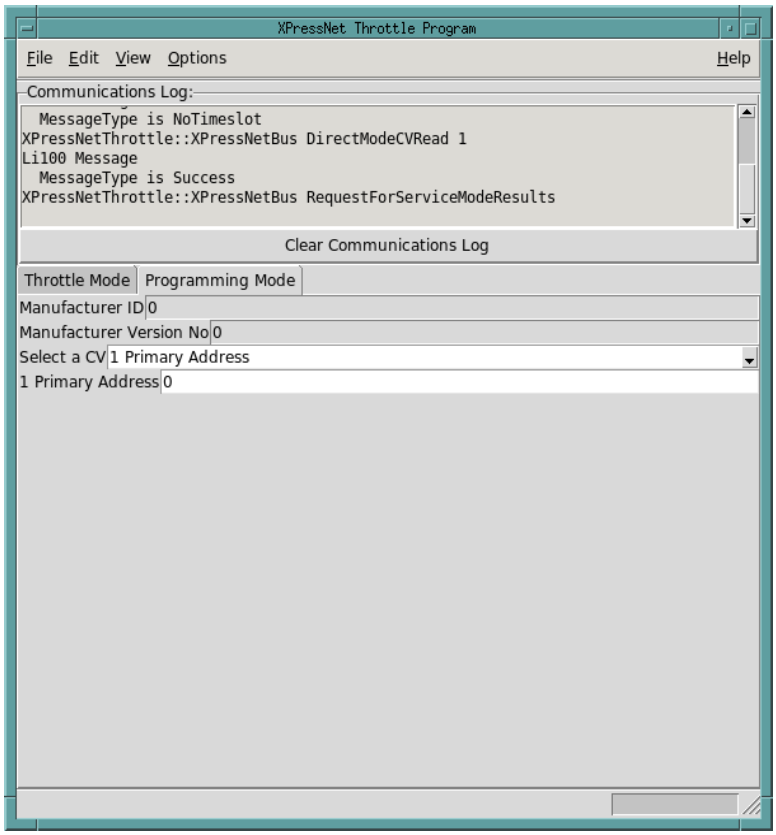


Figure 9.2 XPressNetThrottle Main GUI in Programming Mode

The Manufacturer ID and Version number are fetched and filled in. There is a dropdown menu of standard (common) CVs or you can enter any other CV. The existing value is displayed. You can change it and press ENTER to update the value of the CV register.

9.3 Open Port

The Open Port dialog, shown below, selects the serial port to use to connect to the XPressNet bus.

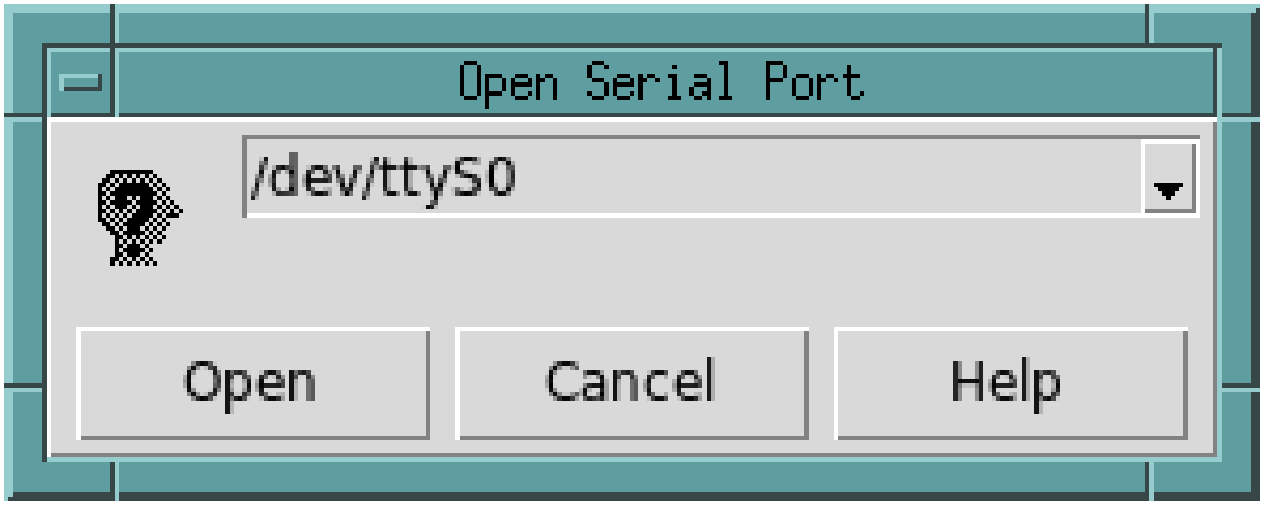


Figure 9.3 XPressNetThrottle Open Port dialog

Chapter 10

Generic Throttle

The GenericThrottle program is a sample program that provides a "virtual" replacement for a hand-held DCC (or DC!) Throttle on your computer screen. It has no "back-end", that is, it does not actually do anything. It is meant as a starting point for writing your own "virtual" throttle (and DCC programming) program.

10.1 Main GUI

Its basic GUI in Throttle Mode is shown here:

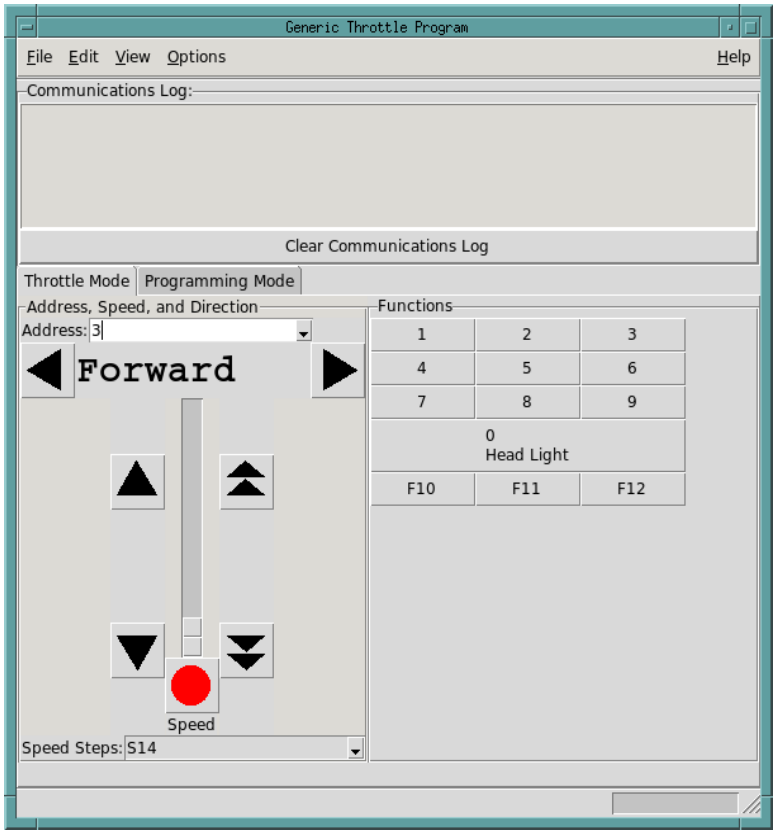


Figure 10.1 GenericThrottle Main GUI in Throttle Mode

On the left is a field to enter the locomotive's address, and buttons for selecting the locomotive's direction and a slider for selecting the locomotive's speed. On the right is an array of buttons to select the locomotive's function bits. By default, the locomotive address is set to 3 but you can enter a different address. The controls are pretty self explainitory.

10.2 Programming Mode

In programming mode, the Main GUI looks like this:

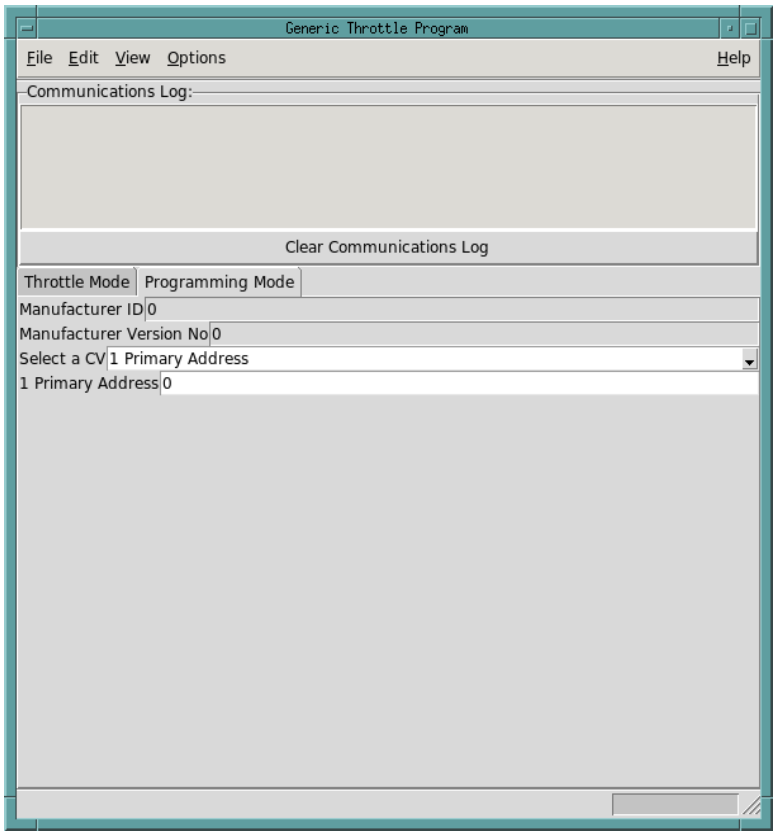


Figure 10.2 GenericThrottle Main GUI in Programming Mode

The Manufacturer ID and Version number are fetched and filled in. There is a dropdown menu of standard (common) CVs or you can enter any other CV. The existing value is displayed. You can change it and press ENTER to update the value of the CV register.

Chapter 11

Time Table (V2) Tutorial

The Time Table is a program designed to create railroad employee timetables. The program's main display is a graph of time (of day) versus distance (along the railroad), gridded at time intervals and at station stops. Trains schedules are represented as colored lines on this graph, with diagonals representing train movement at speed and horizontal lines representing trains "siting" at stations (layovers or switching).

11.1 Creating a new time table

To create an new time table select the `File->New` menu item or the

toolbar button. A "Create a New Time Table" dialog, described in Section Creating a New Time Table. is displayed. This dialog box collects three pieces of information: the name of the new time table, the total time (in minutes) the time table will cover (there are 1440 minutes in a 24 hour day), and the tick interval in minutes. A new time table can also me created from the command line by including the options `-totaltime` and `-timeincrement` along with a name for the new time table.

11.1.1 Creating stations

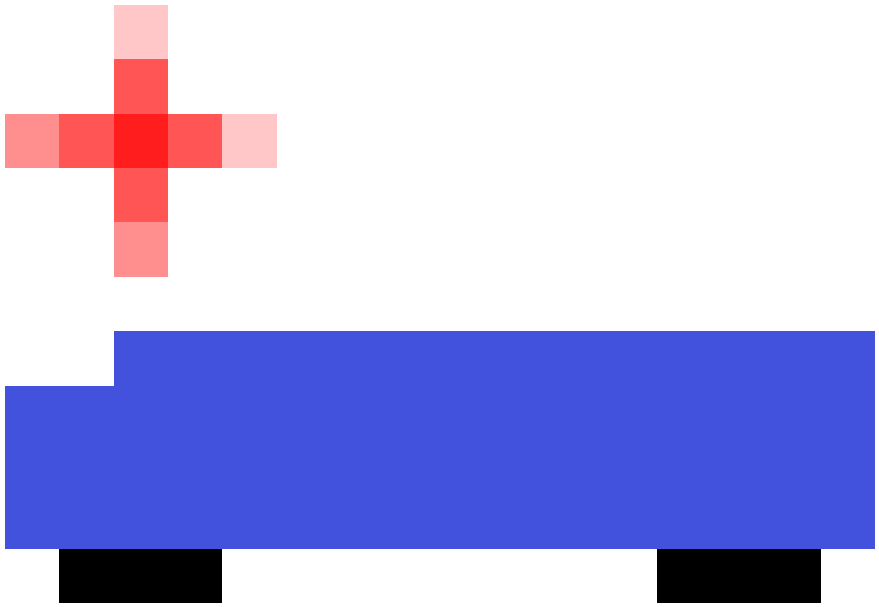
Once the name and the two time elements have been selected, a set of at least two stations need to be created. This is done with the "Create All Stations Dialog", described in Section Creating the station stops for a new time table. This dialog box is used to create stations, which can have zero or more storage tracks. Storage tracks are used when a train has a long layover (and needs to be "out of the way" of other traffic) or when a train terminates and the train set is re-used for a different schedule, generally in the opposite direction. As the stations and their storage tracks are created, they are displayed in the station listing in the upper part of the dialog.

11.1.2 Creating cabs

After creating all of the stations, zero or more cabs can be created. Cabs are mostly for switched block DC layouts, but creating "cabs" for a DCC layout is useful, since it allows for a way to visually group trains operationally. Think of the cabs as a way of defining "crews" (operators). This allows for things like crew (operator) changes as the train moves to different parts of the layout for example.

11.2 Creating trains

Once the stations and cabs have been created, the program displays an empty chart. The chart's `x` axis is time (in minutes). The upper section of the chart has the cabs (if any), the middle part of the chart has the stations, and the bottom part of the chart has the storage tracks (if any). Now we can create a train. This is done by selecting either the `Trains->Add` Train menu item, clicking on the add train (

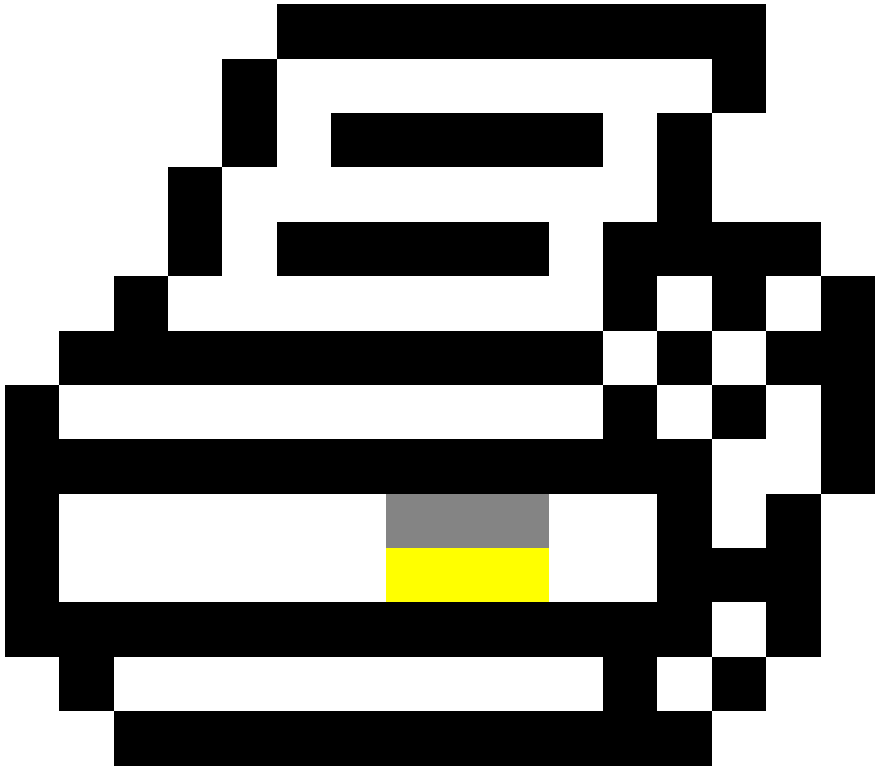


) toolbar button or the `Add a new train` button. All of these display the "Create New Train Dialog", described in Section Create New Train Dialog. Trains have a (common) name, a number (or symbol), a class number, an average speed, a scheduled departure time, and travel between two stations. The train's number (or symbol) needs to be a unique identification of the train. The class is a whole number, with smaller numbers generally being the "higher" class. The class is used to indicate a train's priority and is also used to group similar trains together. The speed is the (scale) speed the train will be traveling between stops. The scheduled departure time is the time the train is scheduled to leave its origin station. The origin and termination stations are the station end points the train travels between. The train will get a "stop" at every intermediate station between these two stations. Note that the train won't be expected to actually stop at any station where the layover time is set to zero. Such stops would just be timekeeping points.

Once the train's basic information is set, the `Schedule` button can be clicked. This shifts to the schedule page, where layovers and cab assignments cab be set. The `Update` buttons propagate the cab settings and adjust the times to allow for the layovers. If the train makes use of station storage tracks, the `Storage` button can be clicked and storage tracks selected. When the train is fully configured, the `Done` button can be clicked to actually create the train.

11.3 Printing a time table

Once all of the trains have been added, it it possible to "print" a timetable. The LaTeX system is used to format the time table and the TimeTable program generates a LaTeX source file (.tex) and will run the LaTeX program, `pdflatex`, to create a PDF file from the LaTeX source file. This process is started with the `File->Print...` menu item or the



toolbar button. This pops up the "Print Dialog", described in Section Print Timetable Dialog. This dialog collects the name of the LaTeX source file, and the path to the LaTeX processing programing, as well as a few other options. It also has a button to configure how the timetable will be formatted.

The `Configure` button pops up the "Print Configuration Dialog", described in Section Print Configuration Dialog, which has three sections, a `General` section which gets some general configuration settings, a `Multi` section for various configuration settings relating to printing multiple tables, and a `Groups` section, for configuring groups of trains. Some of the configuration assumes some knowledge of LaTeX. A visit to the TeX and LaTeX web pages (`http://www.tug.org`) is a good place to start, with the beginner's page at `http://www.tug.org/begin.html` as the obvious starting point. You don't really have to learn how to use LaTeX, you just need to have a TeX/LaTeX system installed. The only other issue is the `TimeTable.sty` file. This file either needs to be installed somewhere in the TeX/LaTeX search path or it needs to be in the same directory as the LaTeX source file generated by the TimeTable program. You will need to learn a little about LaTeX if you want to include various sorts of customizations.

Chapter 12

Time Table (V2) Reference

The Time Table (V2) program is a hybrid program, consisting of a Tcl/Tk GUI on top of a C++ class library. The GUI provides the user interface to the algorithms and data structures contained in the C++ class library. This program was inspired by chapter 8 of the book *How to Operate Your Model Railroad* [1] by Bruce A. Chubb. I strongly recommend reading this chapter fully before using this program. This program implements the methods described in this chapter, in an automated fashion.

12.1 Command Line Usage

There are two formats for the TimeTable program's command line. The command line can either have a single file name, the name of an existing time table file or it can have two options (`-totaltime` and `-timeincrement`) and the name of a new time table. The first form loads an existing time table (see Section Loading an Exiting Time Table File and the second form creates a new time table (see Section Creating a New Time Table. These two command line formats are shown here:

```
TimeTable oldtimetablefile
TimeTable -totaltime time -timeincrement time nameoftimetable
```

12.2 Layout of the Main GUI

The main GUI window is shown here:

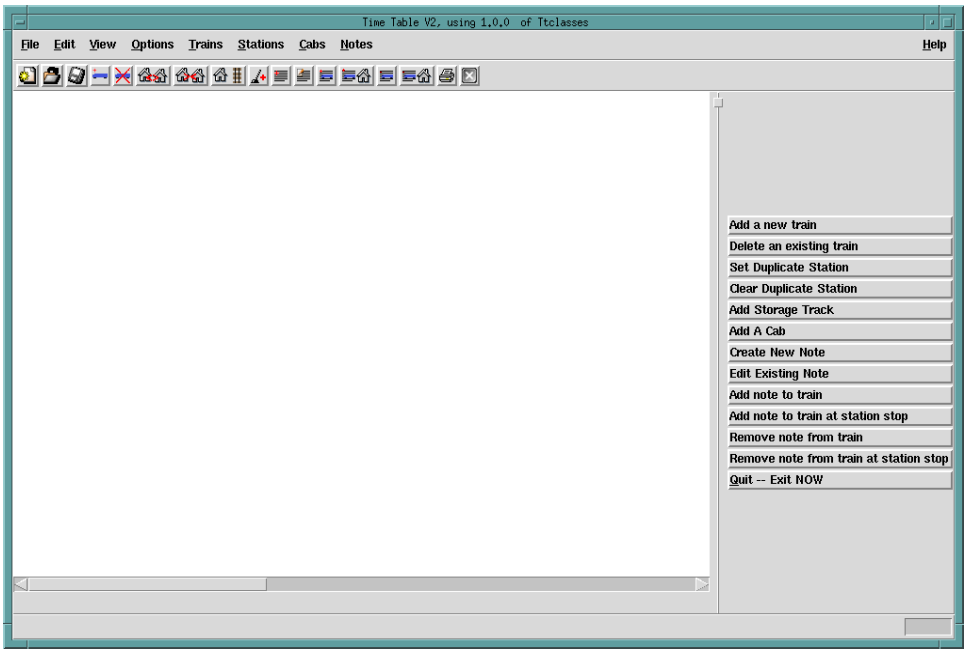


Figure 12.1 The main GUI screen of the Time Table (V2) Program

It contains a menu bar, a toolbar, a time table chart, and a button menu. The toolbar is shown here:

Figure 12.2 The Toolbar of the Time Table (V2) Program

The button menu is shown here:

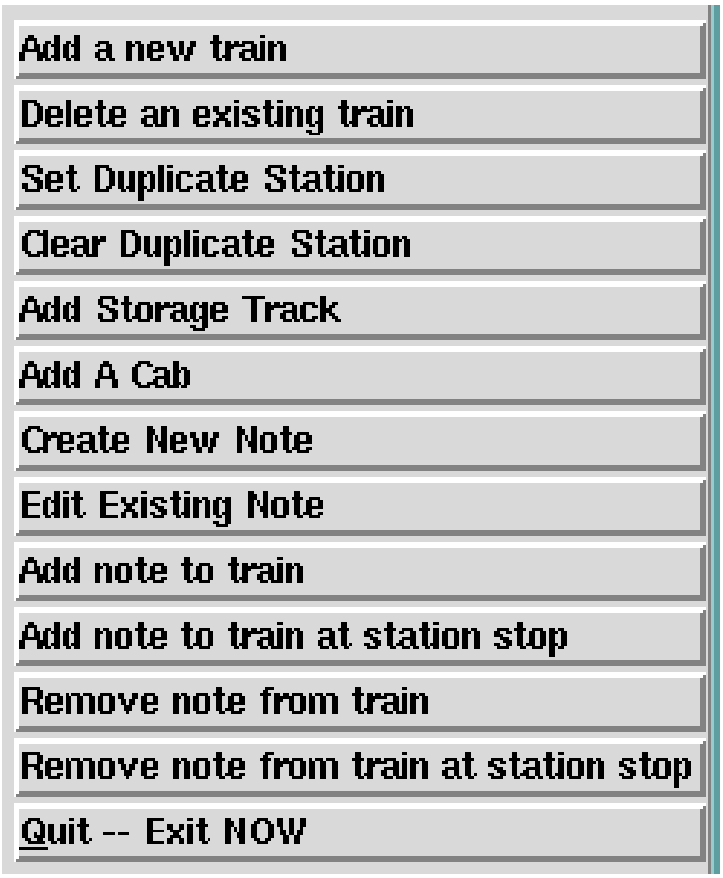


Figure 12.3 The Button Menu of the Time Table (V2) Program

12.3 Creating a New Time Table

Creating a new time table can be done from the command line by specifying a total time (in minutes) value with the `-totaltime` option and a time increment value (in minutes) value with the `-timeincrement` option and a name for the new time table (as shown in the second line above). A new time table can also be created with the `New` menu item of the `File` menu or the

toolbar button. These later two methods use the "Create a New Time Table" dialog, shown below, to get the total time, time increment, and the name of the new time table. If there is a time table file already loaded, a confirmation dialog will be displayed.

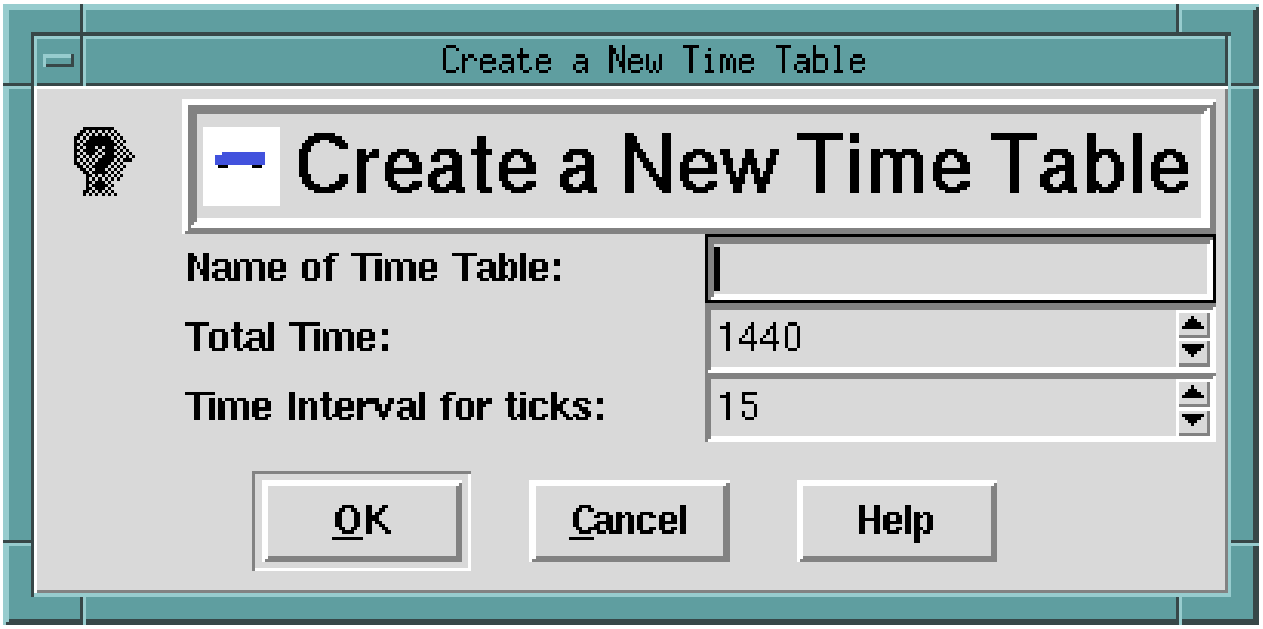


Figure 12.4 Create A New Time Table dialog

A simple chart with three stations, four cabs (labeled "Crew 1" through "Crew 4"), and two storage tracks is shown below.

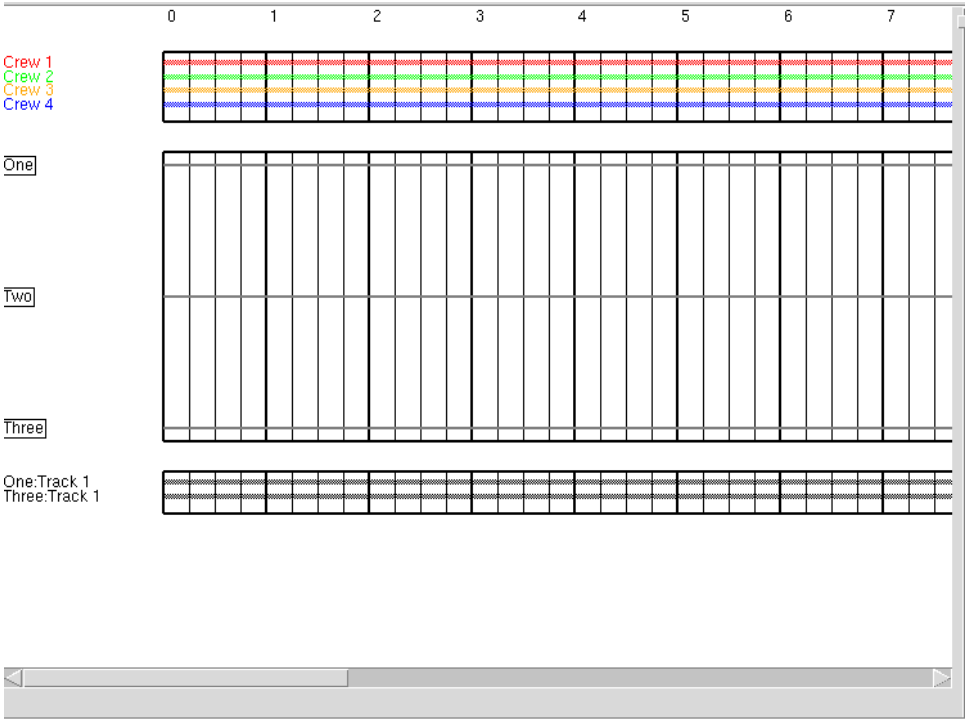


Figure 12.5 Simple chart with three stations, four cabs, and two storage tracks

12.3.1 Creating the station stops for a new time table

Stations for a time table must all be created when the time table is created. Stations cannot be added or removed later. When a new time table is created the "Create All Stations Dialog", shown below, is displayed to create all of the station stops.

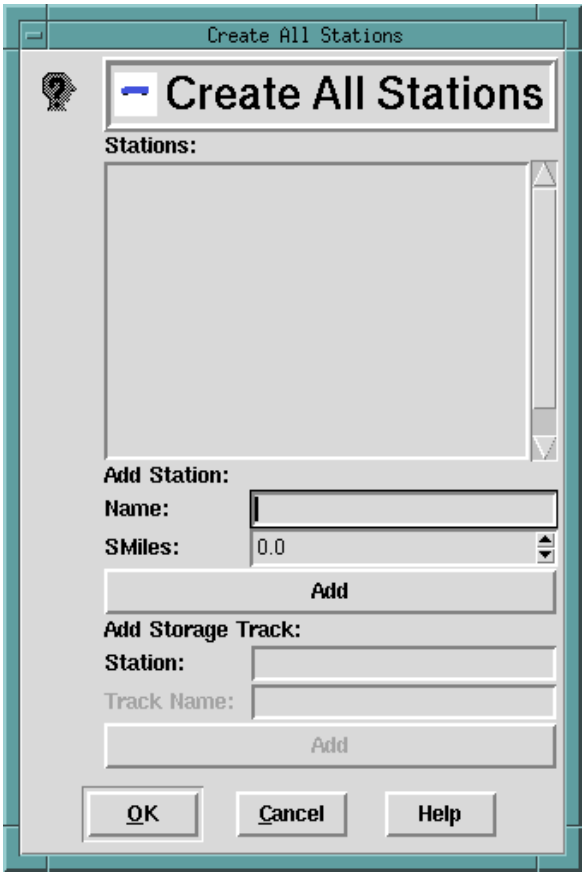


Figure 12.6 Create All Stations Dialog

12.3.2 Create All Cabs Dialog

Once the stations have been created, an initial set of "cabs" can be created. Commonly, cabs are only used on block switch DC layouts, but the cabs can be used as with a DCC layout as a way to associate trains with different operating "crews" (operators) or just to identify different classes of trains by color, etc. The "Create All Cabs" dialog, shown below, is used to bulk create an initial set of cabs.

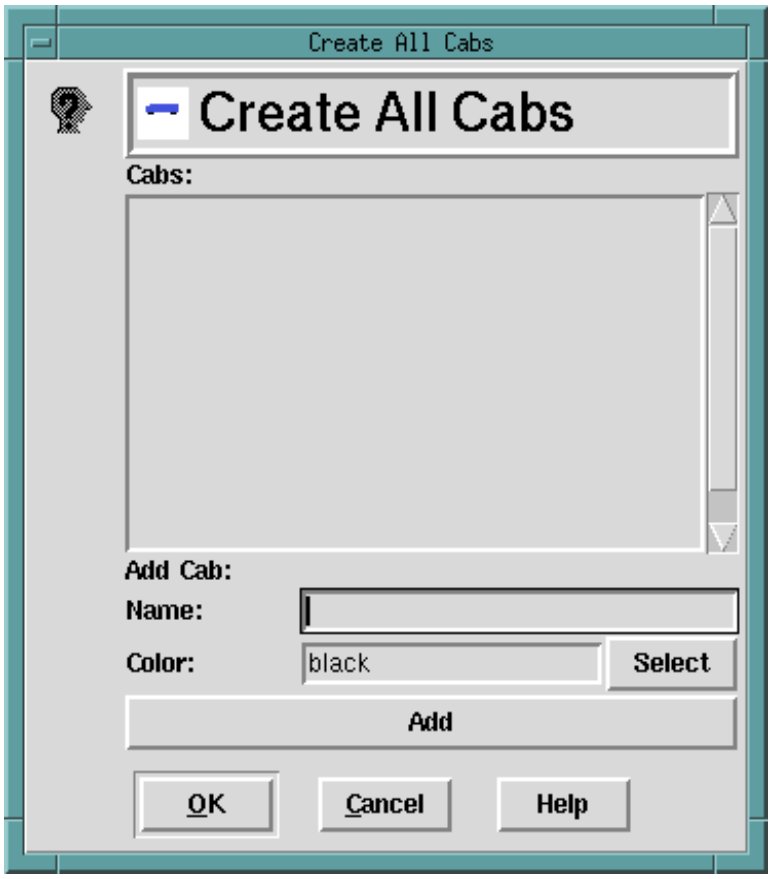


Figure 12.7 Create All Cabs Dialog

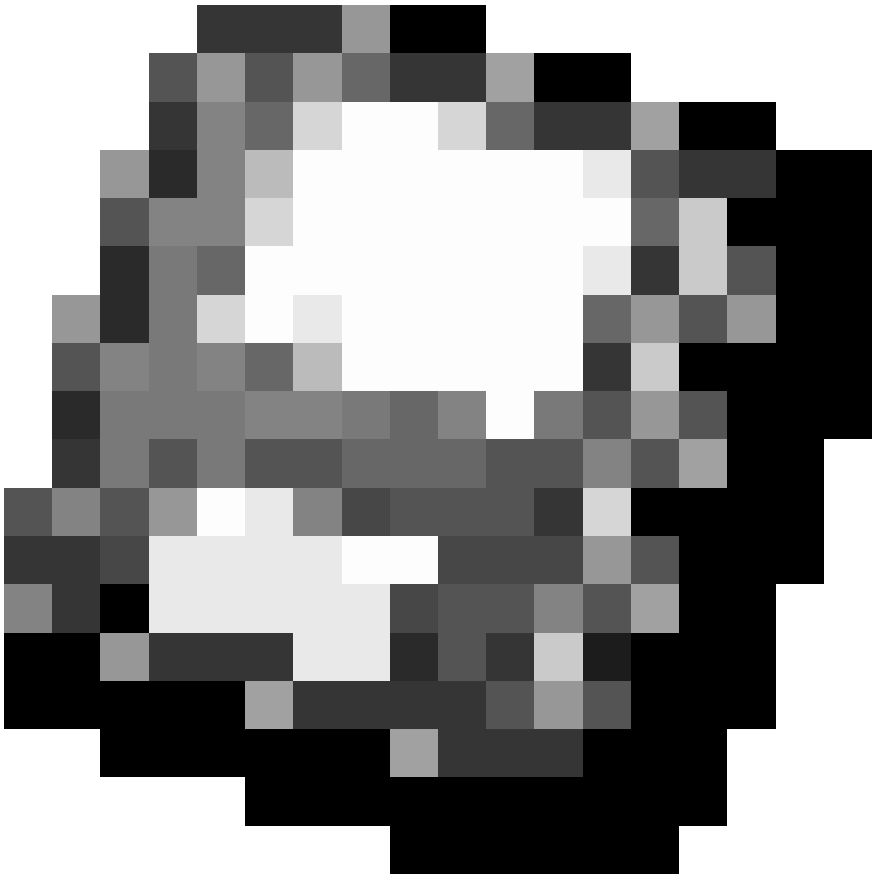
12.4 Loading an Exiting Time Table File

An existing time table file can be loaded from the command line (as shown in the first line of the CLI usage, with the `Open`... menu item of the `File` menu or the

toolbar button. If there is a time table file already loaded, a confirmation dialog will be displayed.

12.5 Saving a Time Table File

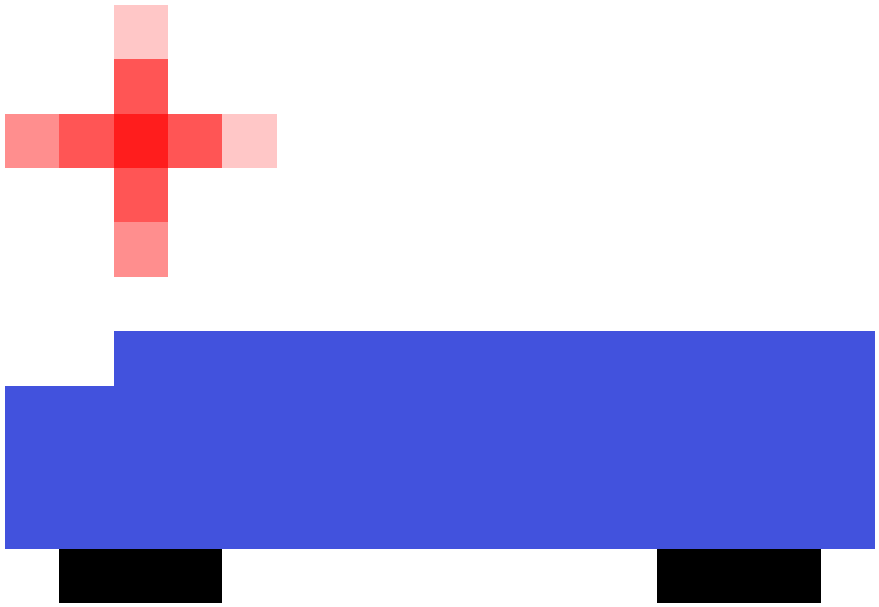
The currently loaded time table can be saved with either the `Save` (or `Save` As...) menu item of the `File` menu or the



toolbar button.

12.6 Adding Trains

Trains are added using the either the `Add Train` menu item of the `Trains` menu, clicking on the add train (



) toolbar button or the `Add a new train` button. All of these display the "Create New Train Dialog", described in Section Create New Train Dialog.

12.6.1 Create New Train Dialog

The "Create New Train Dialog" first collects some basic information about the new train, as shown below. The basic train information consists of the train's common name, its number (or symbol), its class number, its average speed, its scheduled departure time, and the two stations it travels between.

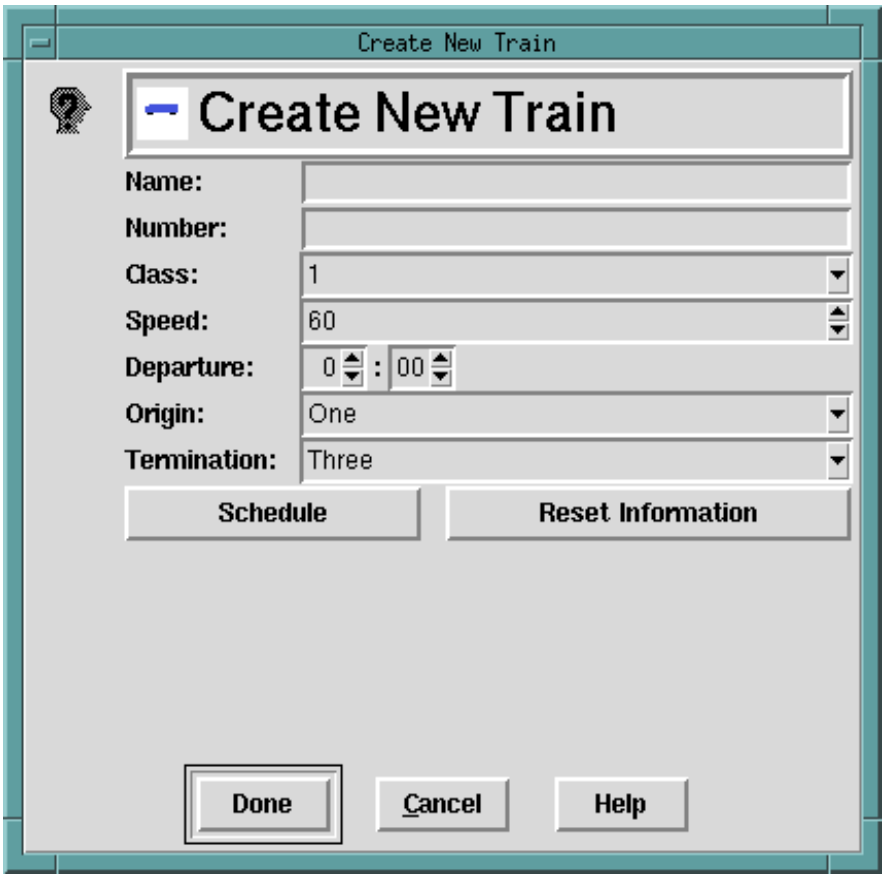


Figure 12.8 Creating a new train dialog, basic information

The train's number (or symbol) needs to be a unique identification of the train. The common name need not be unique. The class is a whole number, with smaller numbers generally being the "higher" class. The class is used to indicate a train's priority and is also used to group similar trains together. The speed is the (scale) speed the train will be traveling between stops. The scheduled departure time is the time the train is scheduled to leave its origin station. The origin and termination stations are the station end points the train travels between.

The `Schedule` button selects the scheduling page of the "Create a @addindex "train, adding a schedule" New Train Dialog", as shown below. On this page, the cab can be selected and layover periods at intermediate stations can be set. The `Update` buttons propagate the cab settings and adjust the times to allow for the layovers.

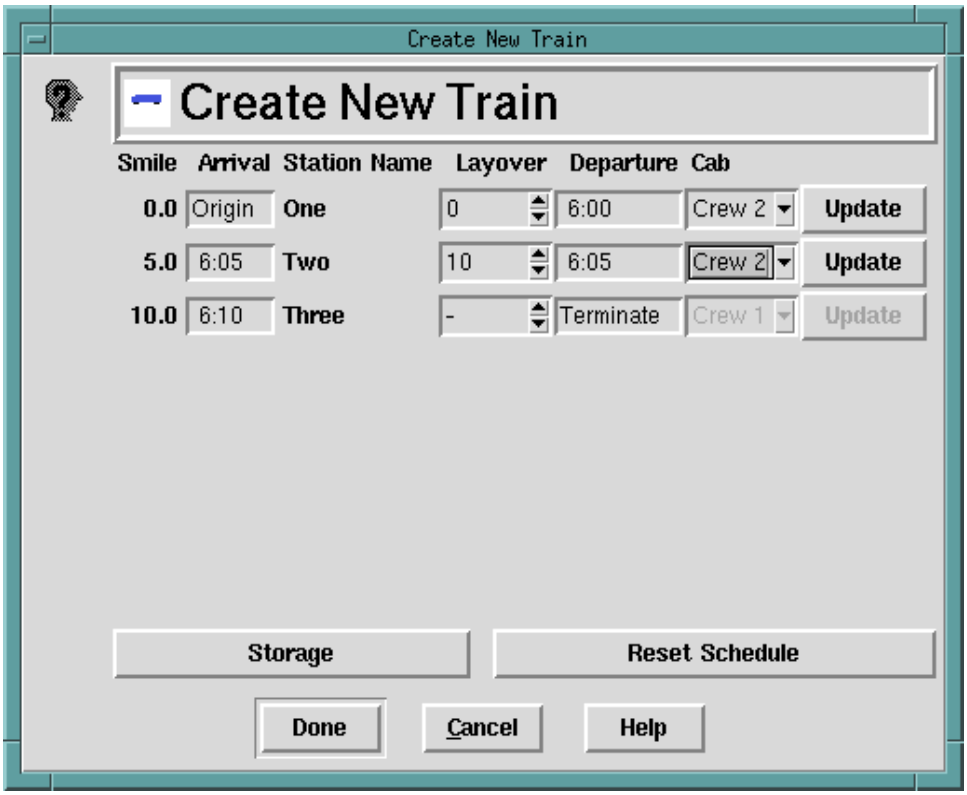


Figure 12.9 Creating a new train dialog, scheduling information

The `Storage` button selects the storage track allocation page of the "Create a New Train Dialog", as shown below. This page lists those stations that have storage tracks available. It only makes sense to select storage tracks for intermediate stops if there is a layover or for originating or terminating stops.

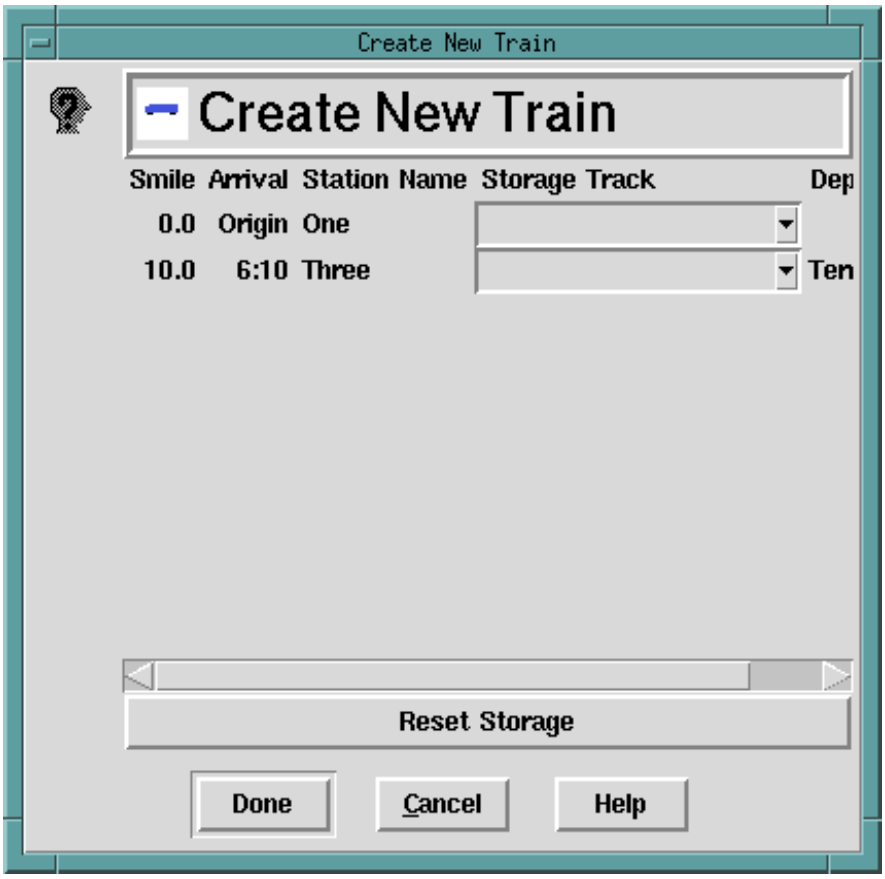
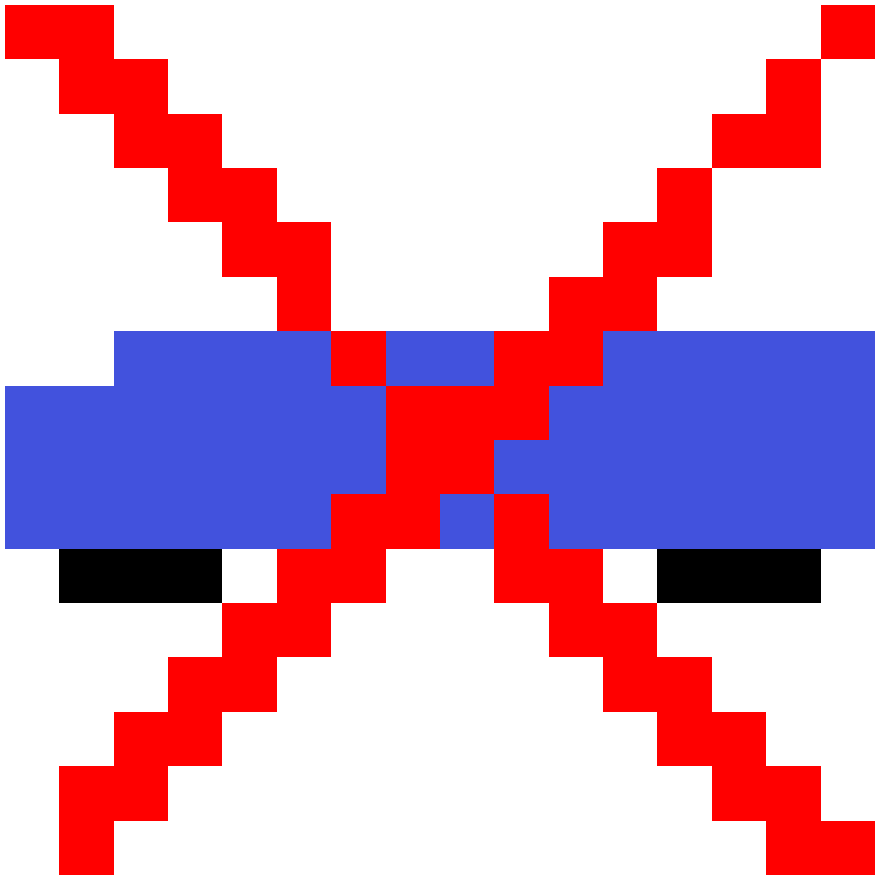


Figure 12.10 Creating a new train dialog, storage track selection

12.7 Deleting Trains

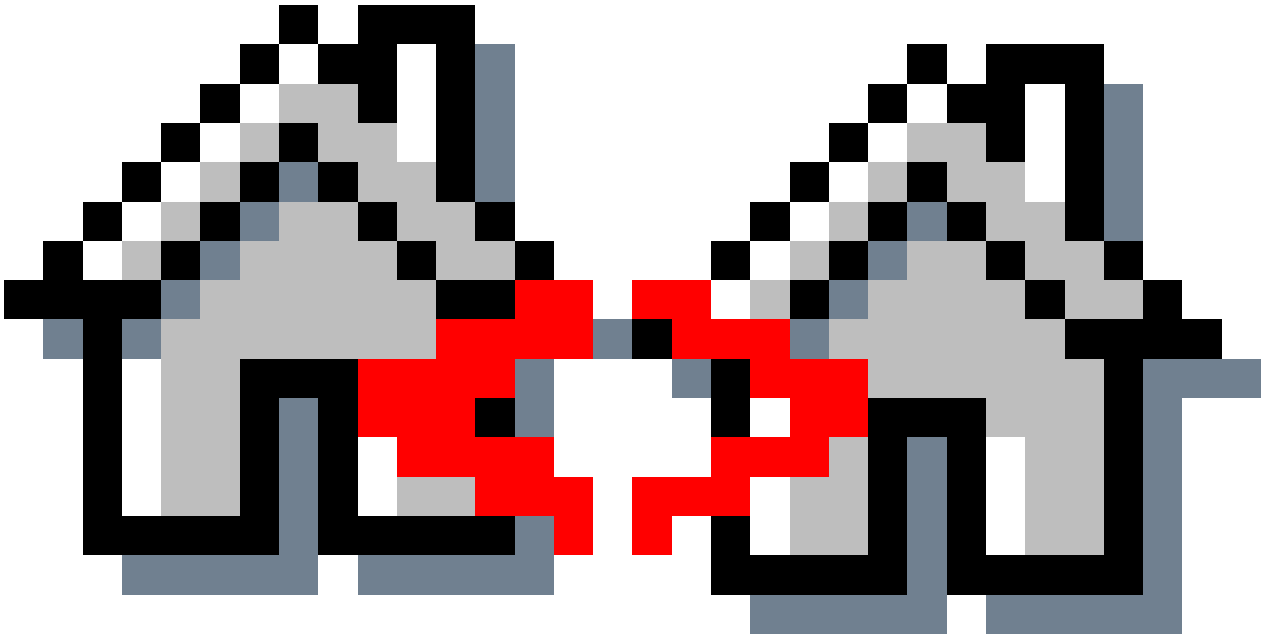
Trains are deleted using the `Delete Train` menu item of the `Trains` menu, clicking on the delete train (



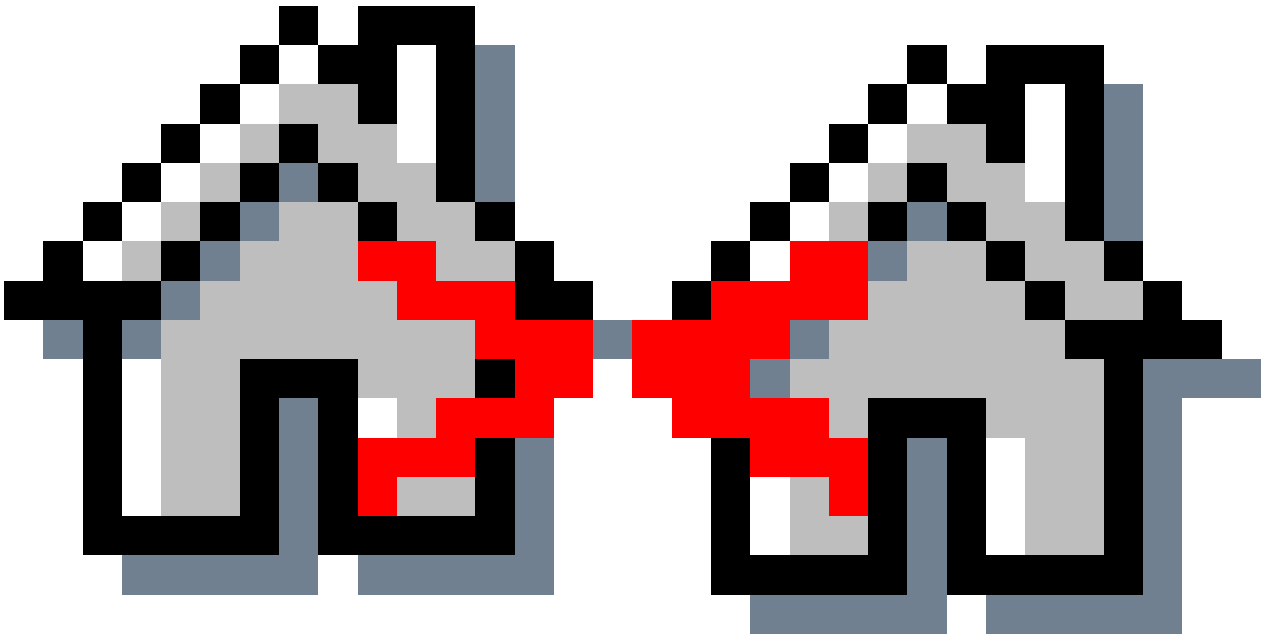
) toolbar button or the `Delete an Existing train` button. All of these display the "Select One Train Dialog", described in Section Select One Train Dialog. A delete confirmation dialog will also be displayed.

12.8 Linking and Unlinking Duplicate Stations

Duplicate stations occur mostly with "out and back" type layouts where the opposite ends of the line are modeled with the same trackage (usually a yard). Duplicate stations also occur with reverse loops. In all cases, these are stations which are logically different, but which use the same tracks. There is an example in Figure 8-4 on page 86 of [1]. It is necessary to keep track of this trackage in the schedule. The duplicate station linking handles this. Duplicate stations need to be setup before trains have been added. The `Set Duplicate Station` and `Clear Duplicate Station` menu items of the `Stations` menu, the



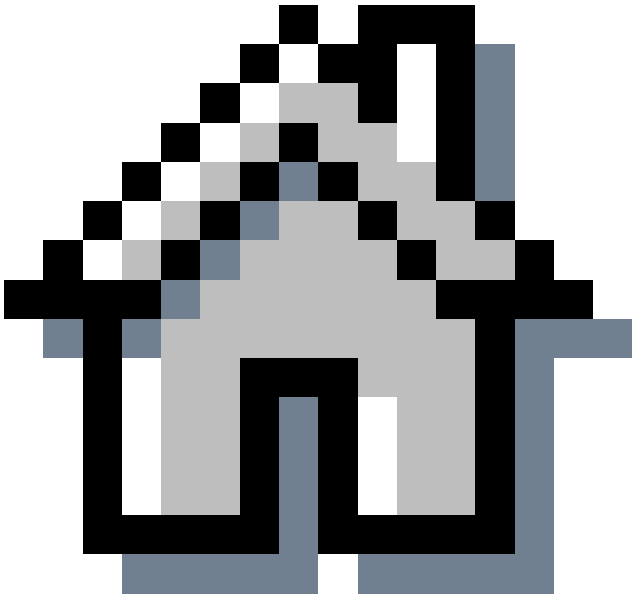
and

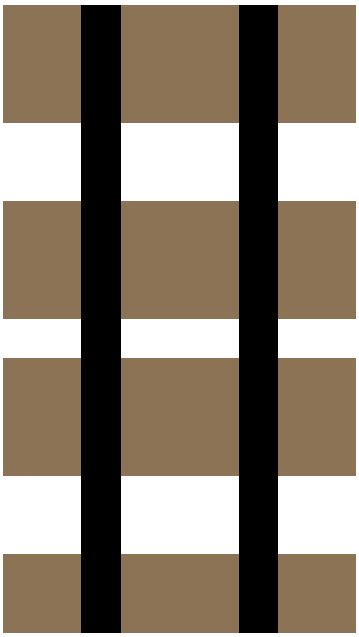


toolbar buttons, and the `Set Duplicate Station` and `Clear Duplicate Station` buttons set and clear duplicate stations.

12.9 Adding Station Storage Tracks

Storage tracks are sidings where whole trains can be stored, either during a long layover or between trips. The `Add Storage Track` menu item of the `Stations` menu, the

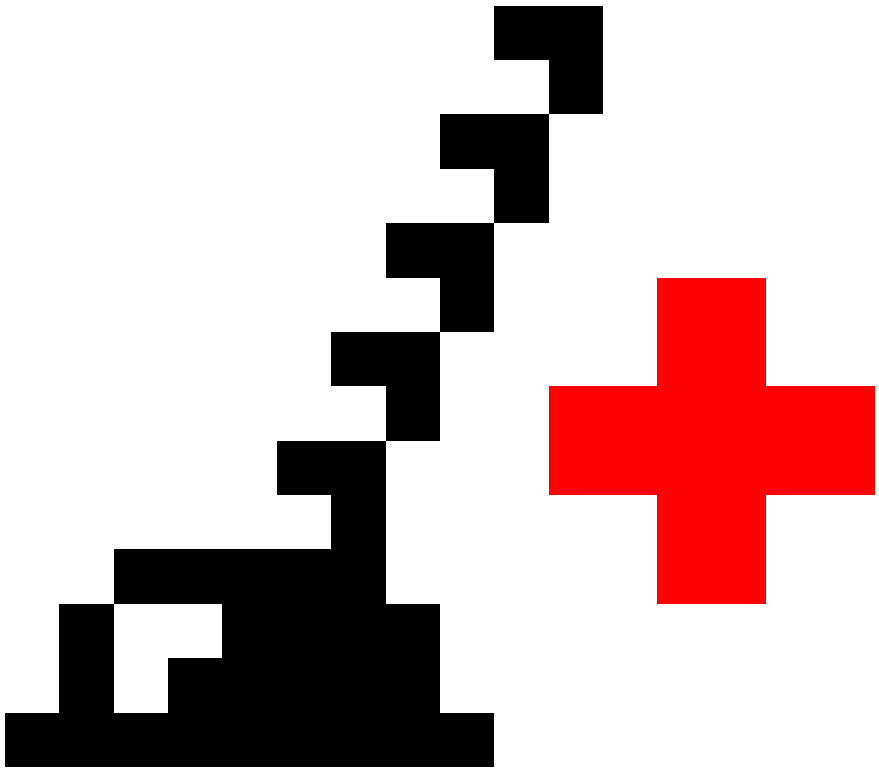




toolbar button, or the `Add Storage Track` button are used to add a storage track to a station.

12.10 Adding Cabs

Generally "Cabs" refer to the separate throttle controls on a block switched DC layout. They are generally non-existent with a DCC layout, but virtual cabs might be used as a way of assigning crews (operators) to a train or to a segment of a train's run. Cabs are added with the `Add A Cab` menu item of the `Cabs` menu, the



toolbar button or the `Add A Cab` button.

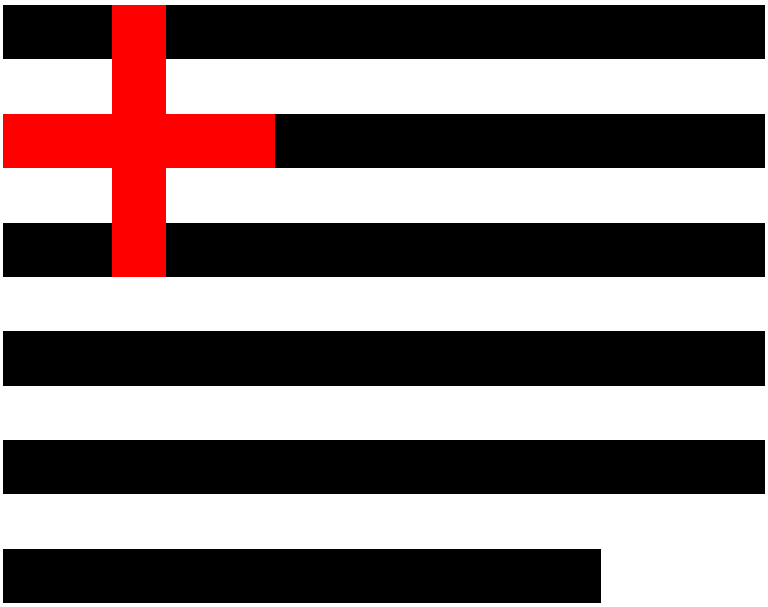
12.11 Handling Notes

Notes are brief memos about the operating rules in effect. There is a

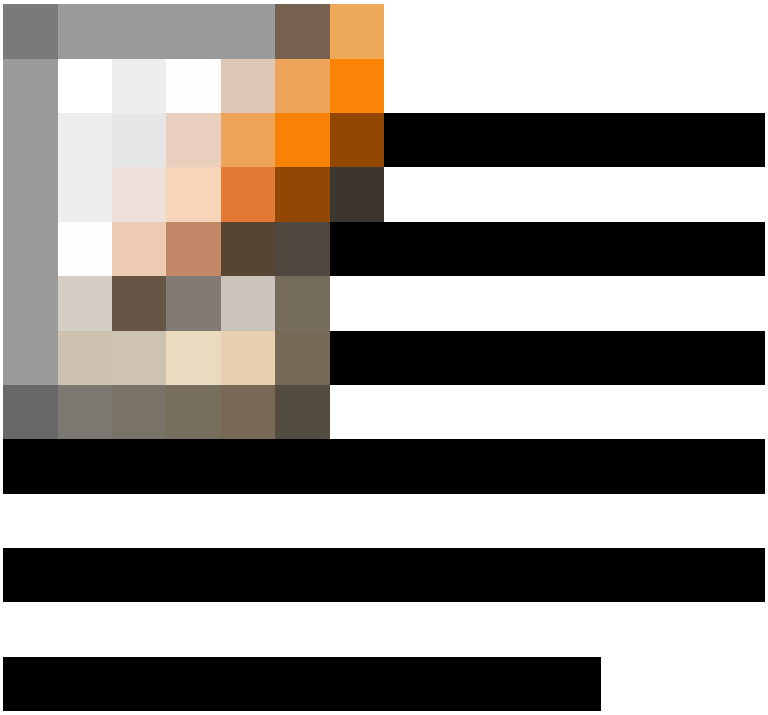
- single pool of notes. Notes from this pool can be associated either with a whole train or with a train at a station stop. The notes can specify schedule exceptions (eg "Daily except Saturdays, Sundays, and Holidays"), or operating rules relating to meets.

12.11.1 Creating New Notes and Editing Existing Notes

Notes are created and edited the `Create New Note` and `Edit Existing Note` menu items of the `Notes` menu, the



and



toolbar buttons, or the `Create New Note` and `Edit Existing Note` buttons. The the "Note editor dialog", shown below is used to create or edit the note. Notes are numbered consecutively starting with 1.

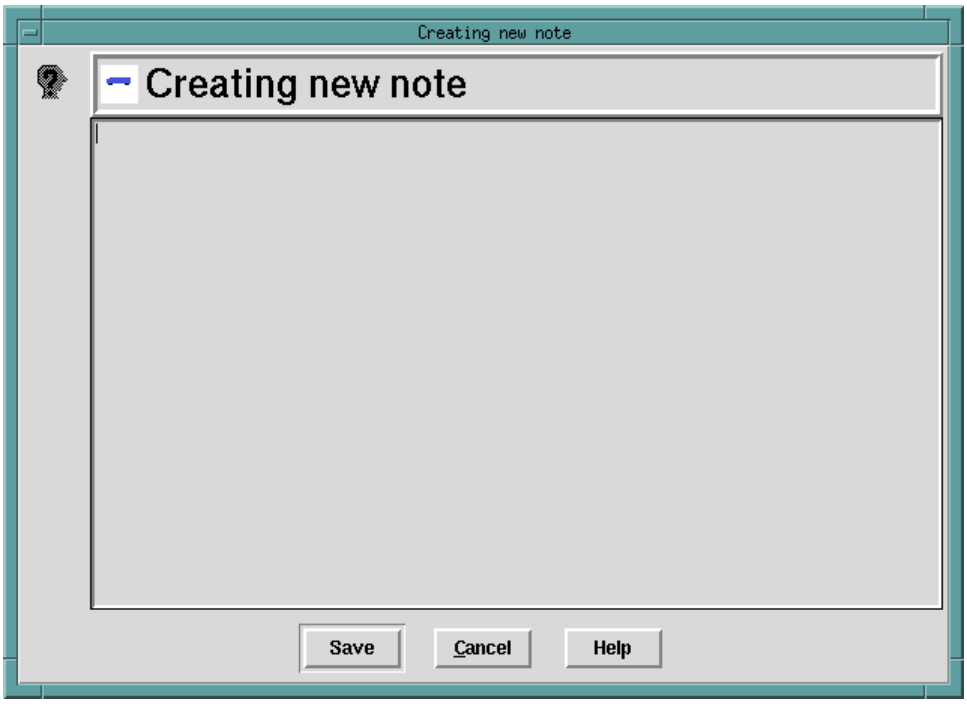
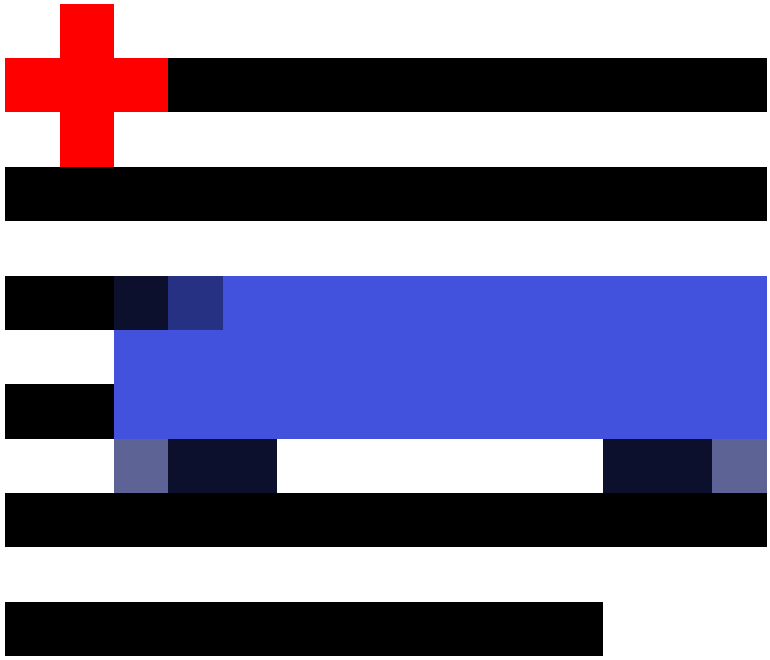


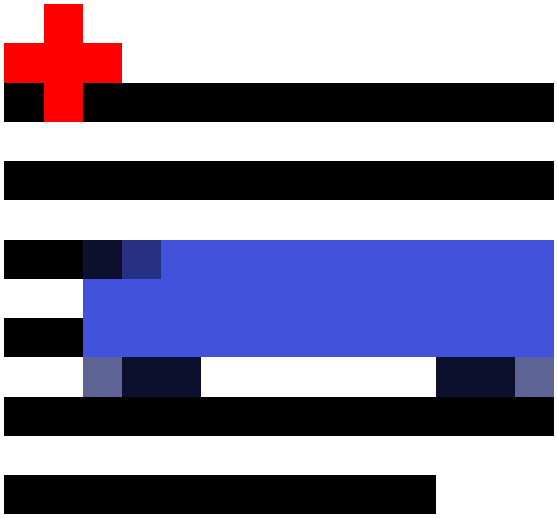
Figure 12.11 Note editor dialog

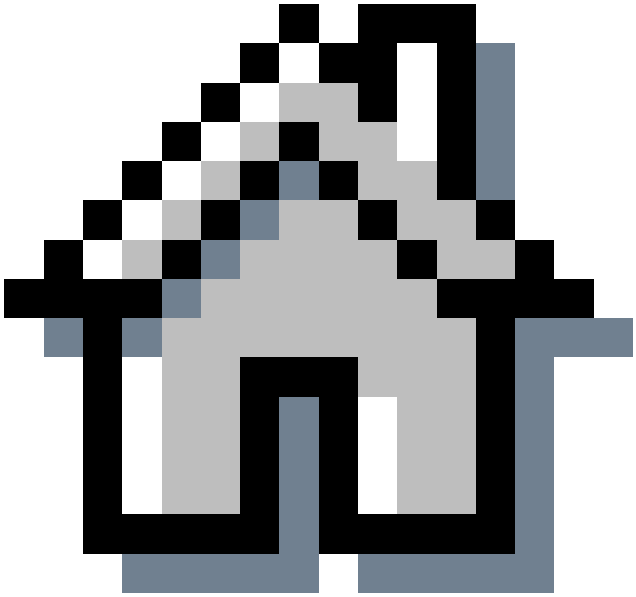
12.11.2 Adding and Removing a Notes To Trains

Notes are added to trains or removed from trains with `Notes` menu items `Add note to train`, `Add note to train at station stop`, `Remove note from train`, and `Remove note from train at station stop`; the

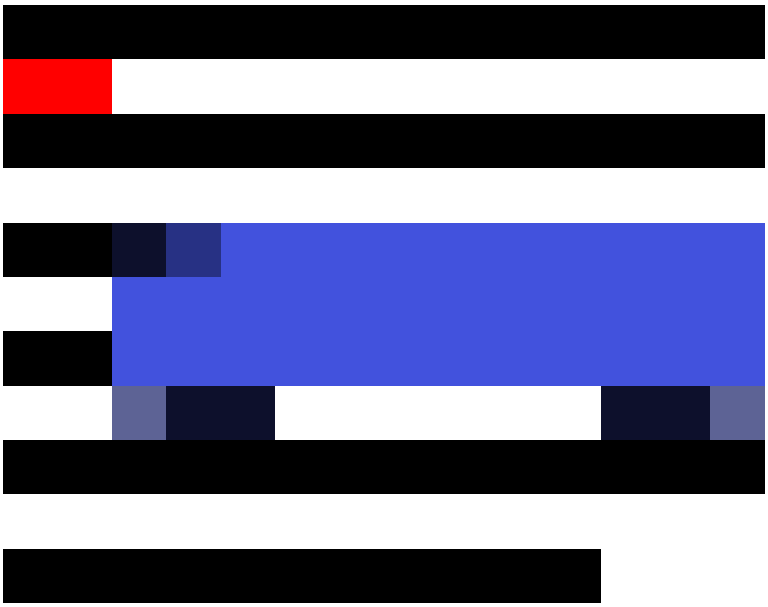


,

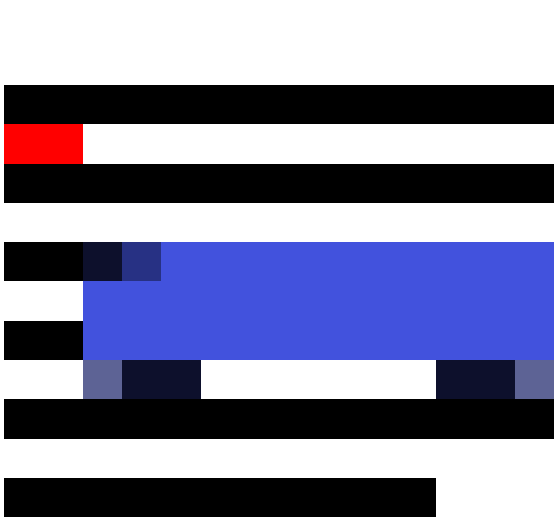


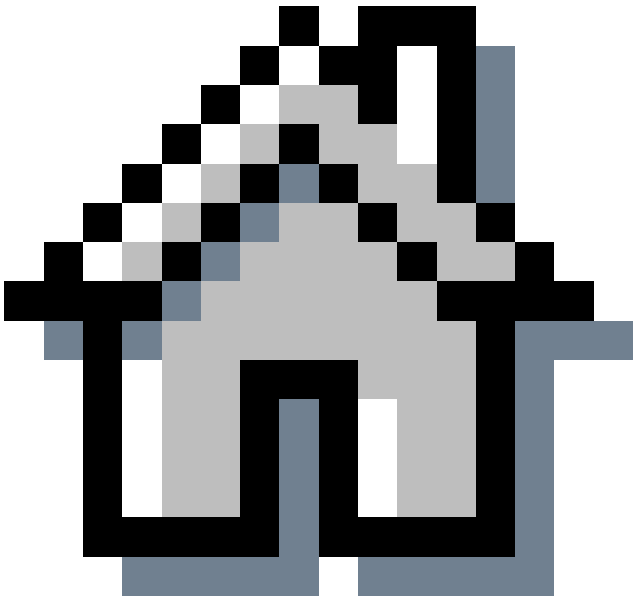


,



, and





; or the Add note to train, Add note to train at station stop, Remove note from train, and Remove note from train at station stop buttons. All of these display the "Add (or Remove) Note dialog", shown below.

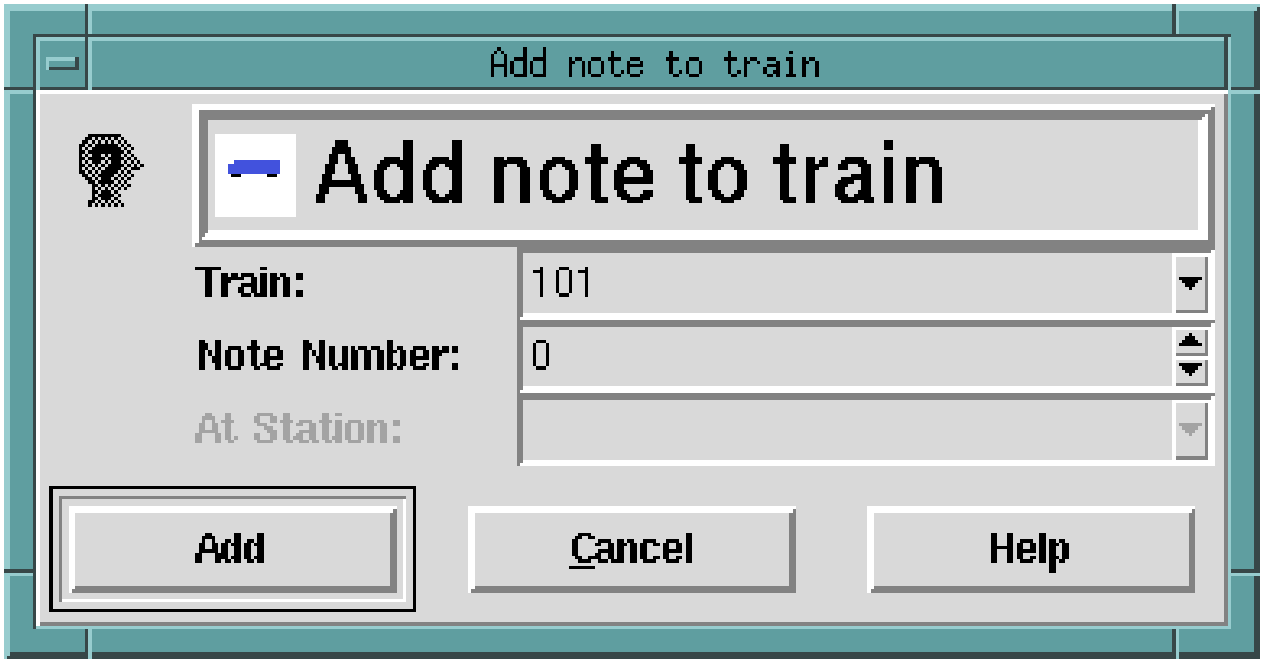


Figure 12.12 Add (or Remove) Note dialog

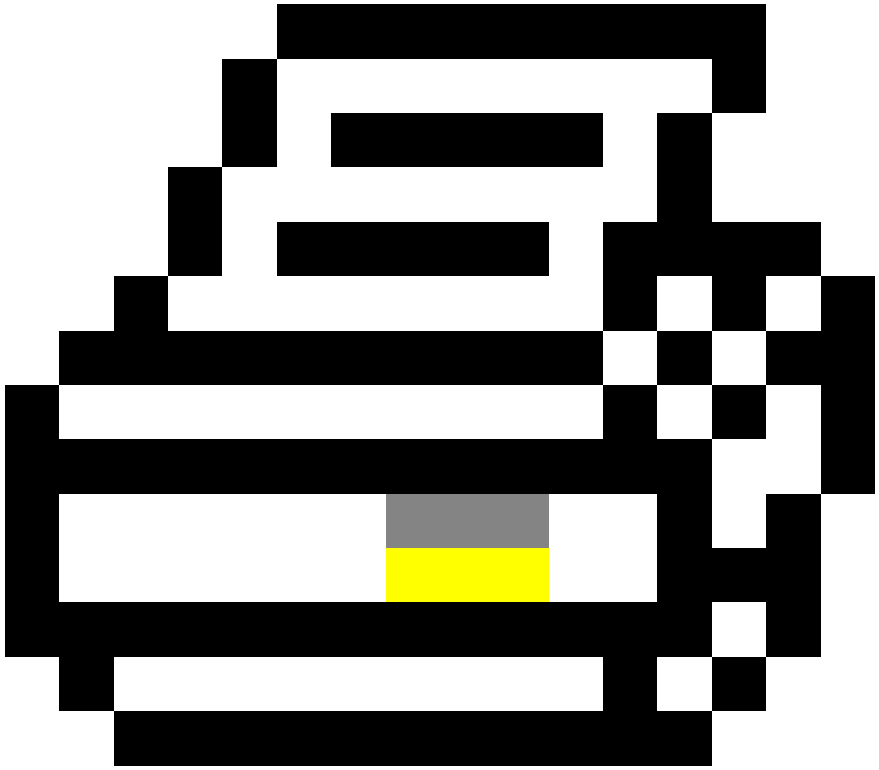
12.12 Printing a Time Table

"Printing" a time table actually means creating a LaTeX file and then processing that LaTeX file through a LaTeX processing program (typically pdflatex). LaTeX provides the means to produce a professionally formatted document and has the means to provide things like table of contents and the creation of a final document in a selection of different final formats, including PDF (via pdflatex), PostScript (via latex and dvips) or HTML (via the htlatex script from tex4ht package).

Much of the formatting is customizable through the insertion of LaTeX code fragments as well as through various parameter settings. It is also possible to edit the LaTeX style file that comes with the Time Table program (Time↩ Table.sty) to tweak some of the fine details of the formatting as well. [1]

The Print menu item of the File menu or the

[1] Some knowledge of how LaTeX works is recommended when messing with the style file.



toolbar button initiate the print process by displaying the "Print Timetable" dialog, described in Section Print Timetable Dialog.

12.12.1 Print Timetable Dialog

The "Print Timetable" dialog, shown below, collects the basic information needed to generate and process a LaTeX source file from the time table data structure. This information consists of the name of the name of the LaTeX source file to create, the LaTeX processing program (`pdflatex` by default), whether to run the LaTeX processing three times (to get the table of contents right), the name of any post processing command (such as `dvips` if using plain `latex`). Most of the time, this is enough for a standard, basic time table. The `Configure` button can be used to configure a selection of options using a "Print Configuration" dialog, described in Section Print Configuration Dialog.

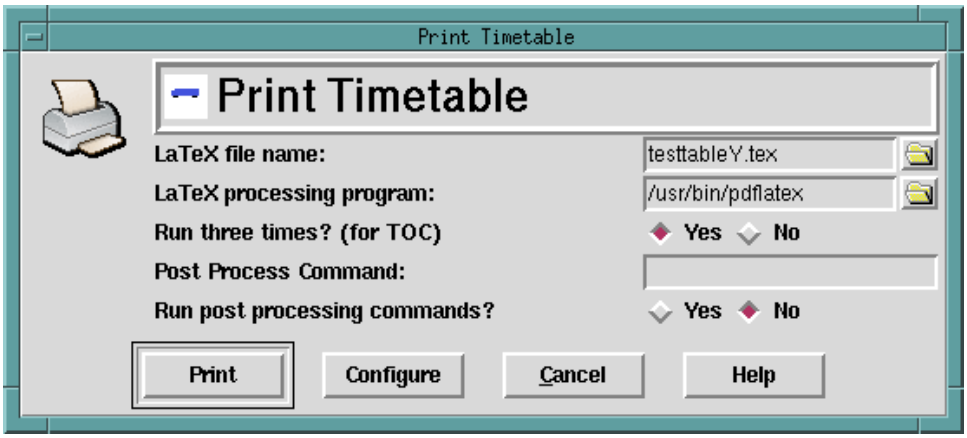


Figure 12.13 Print Timetable dialog

Once the settings and configuration have been set, the `Print` initiates the process. First a LaTeX source file is generated, then the LaTeX processing program is run once or three times. The output from these runs are displayed in a process log window (LaTeX outputs a fair amount of diagnostic output, most of which can be ignored). If you are using the default processor (`pdflatex`), you should now have a PDF file which can be viewed or printed with the PDF viewer of your choice.

12.12.2 Print Configuration Dialog

The Print Configuration Dialog, shown below, provide for the setting of many print configuration options. The general settings, provide for setting the title, subtitle, the date, whether to have LaTeX format for double sided printing, setting the time format, setting the logical direction of trains, column widths, and including additional commands in the LaTeX preamble (usually including additional style packages [2] and style settings). The multi-table settings, provide for settings relating to time tables using multiple tables. These settings include whether to create a table of contents, whether to use multiple tables at all, LaTeX code to precede the table of contents, LaTeX code to precede notes section, the header to use if a single "All Trains" table is generated, and LaTeX code to precede this single "All Trains" table. The groups settings, provide for settings for each group. This includes whether to group by class or to manually group trains and provides for setting the class or group heading and for LaTeX code to precede the group table, and if grouping manually, selecting the trains in the group.

[2] The style pages supertabular and graphicx are already included.

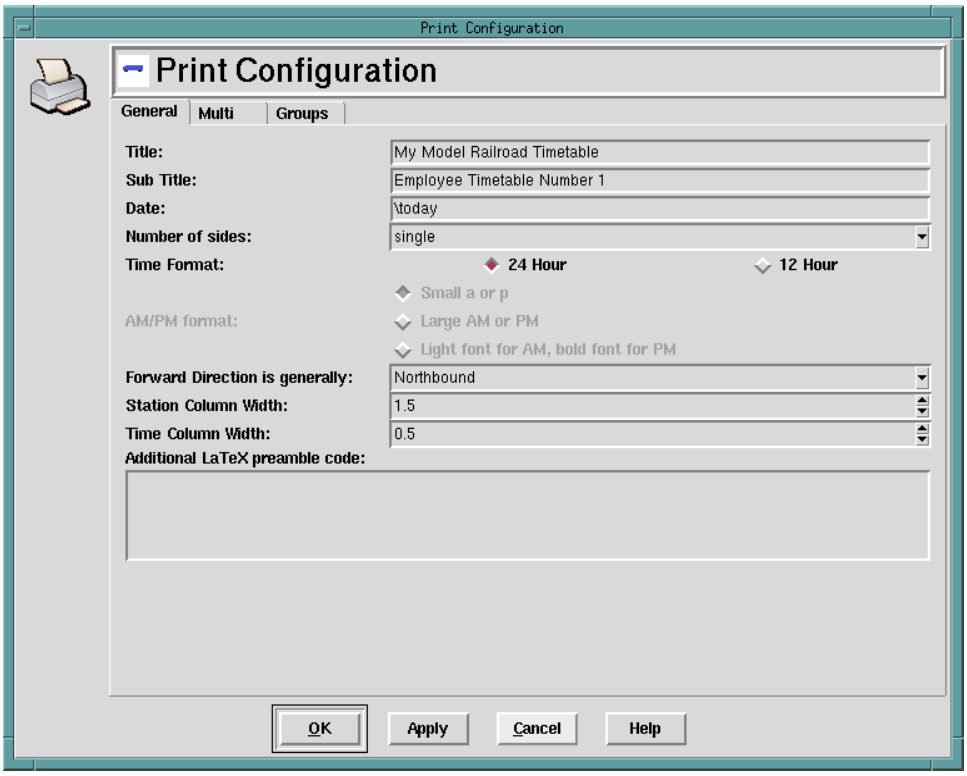


Figure 12.14 Print Configuration dialog, General settings

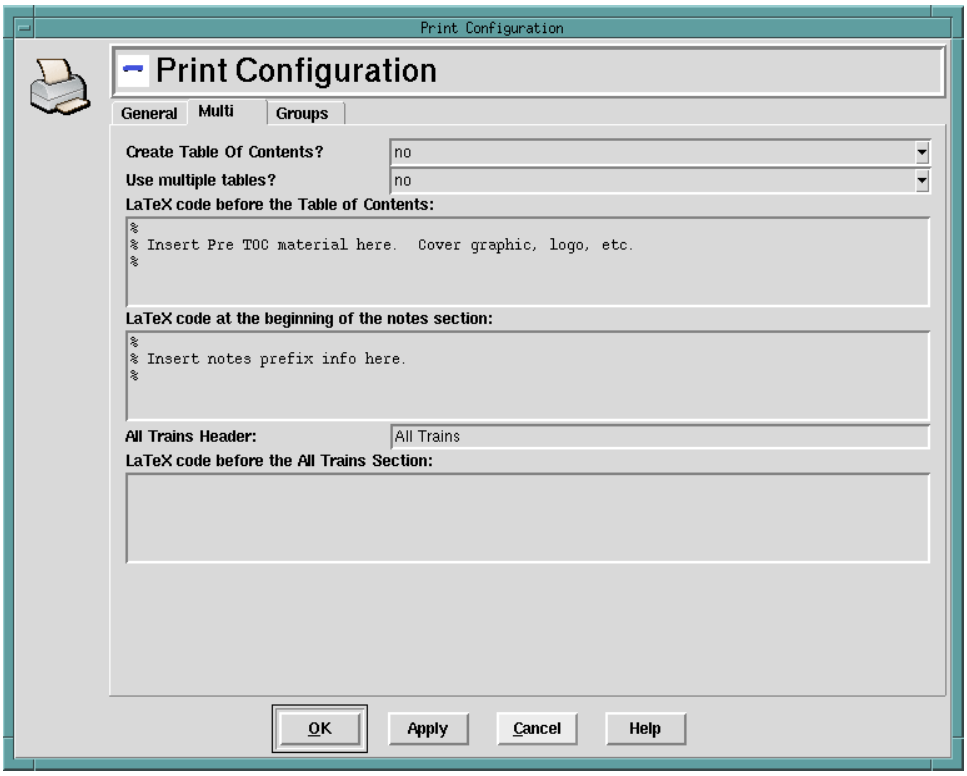


Figure 12.15 Print Configuration dialog, Multi settings

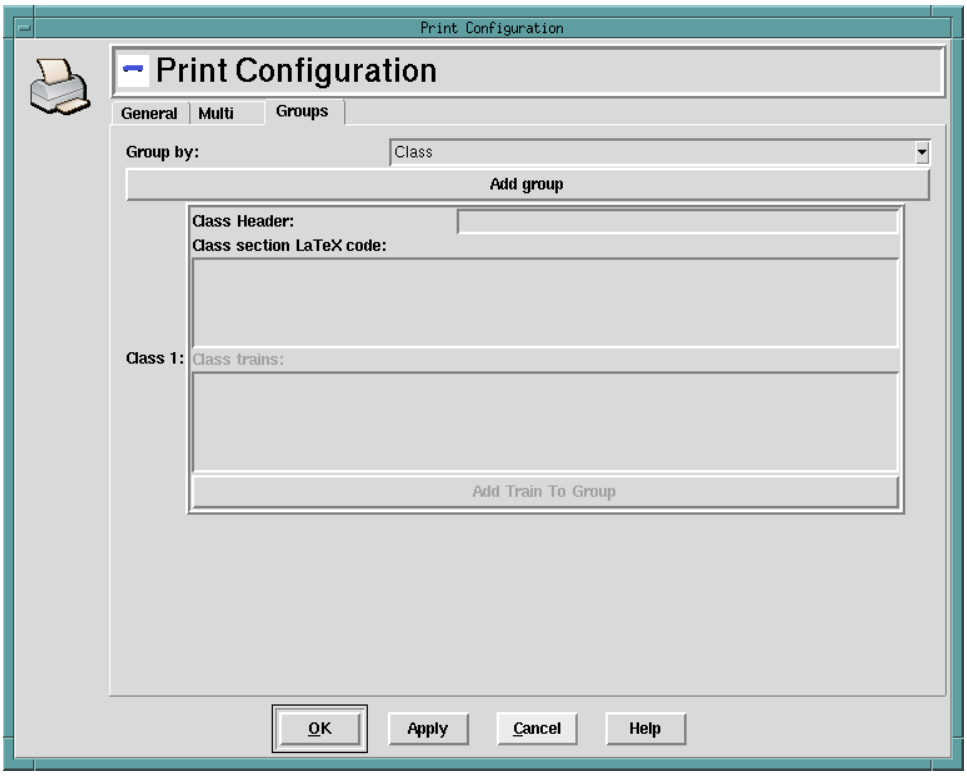
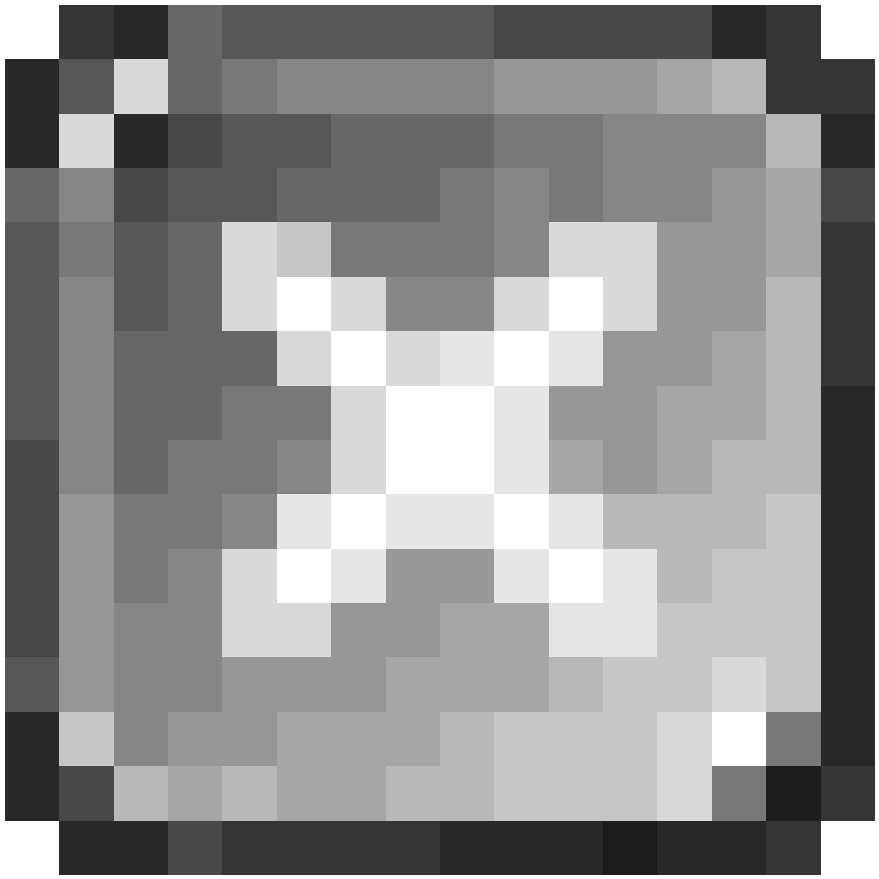


Figure 12.16 Print Configuration dialog, Groups settings

12.13 Exiting From the Program

The `Exit` (or `Close`) menu item of the `File` menu, the



toolbar button, or the `Quit - Exit NOW` button exit the program. A confirmation dialog is displayed to get confirmation.

12.14 Select One Train Dialog

The "Select One Train dialog", shown below, is used to select a train either for deletion (Section Deleting Trains) or for viewing (Section Trains).

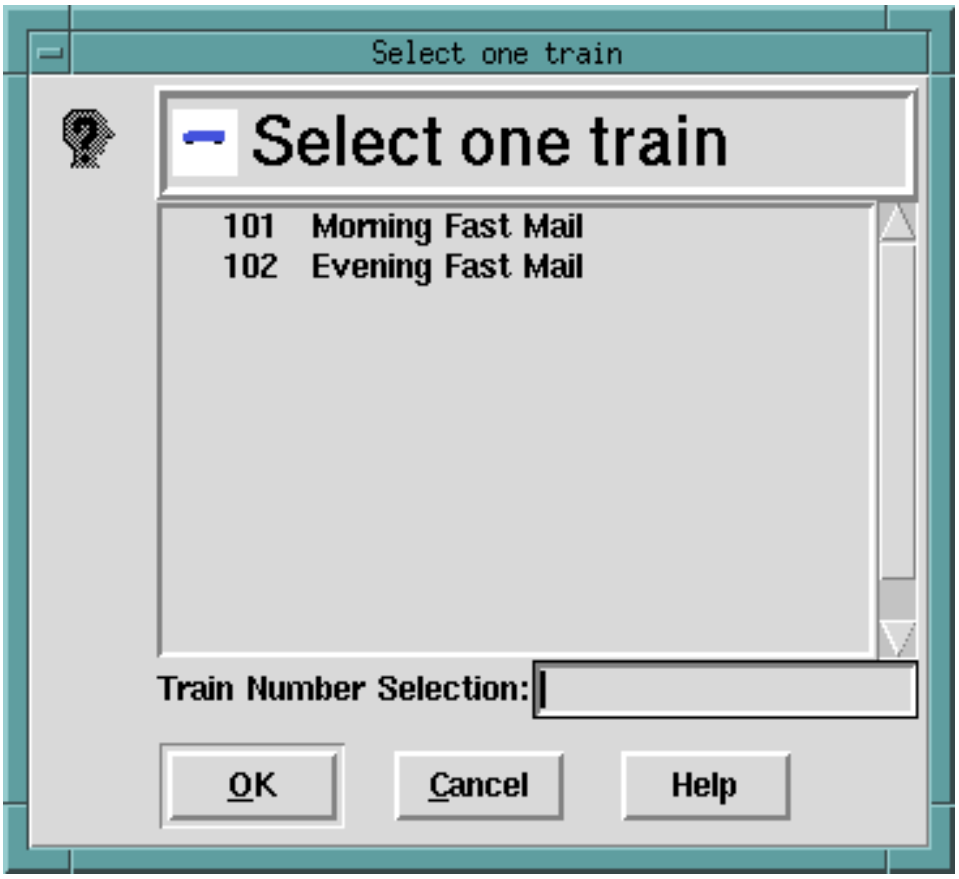


Figure 12.17 Select One Train dialog

12.15 The View Menu

The view menu contains menu items for viewing detailed information about various things, including trains (Section Trains, stations (Section Stations), and notes (Section Notes).

12.15.1 Trains

There are two menu items for viewing trains, `View One Train` and `View All Trains`. The `View One Train` uses the "Select One Train dialog" (Section Select One Train Dialog) to select a train to display detailed information about and the `View All Trains` menu item displays a dialog listing all of the trains, by number and name, with buttons to get more detailed information.

12.15.2 Stations

There are two menu items for viewing stations, `View One Station` and `View All Stations`. The `View One Station` uses the "Select One Station dialog" to select a station to display detailed information about and the `View All Stations` menu item displays a dialog listing all of the stations, by name and scale mile, with buttons to get more detailed information.

12.15.3 Notes

There are two menu items for viewing notes, `View One Note` and `View All Notes`. The `View One Note` uses the "Select One Note dialog" to select a note to display detailed information about and the `View All Notes` menu item displays a dialog listing all of the notes, by number and beginning text, with buttons to get more detailed information.

12.16 System Configuration

The Time Table program has a small number of global configuration options. These are stored in a file named .timeTable (`TimeTable.rc` under MS-Windows) in the current user's HOME directory. These configuration options are:

Path to pdflatex The pathname to the `pdflatex` executable.

Label Width in Chart The width in pixels of cab, station, and storage track labels in the time table chart.

Height of main window The initial height of the main window.

Width of main window The initial width of the main window.

The system configuration file is read at program start up. If the configuration does not exist, a default one is created the first time the program is run.

The `Options` menu manages the system configuration, with menu items to edit the system configuration, save it and reload it.

12.17 Add Cab Dialog

12.18 Add Remove Note Dialog

12.19 Edit Note Dialog

12.20 Edit System Configuration

12.21 Edit Train Dialog

12.22 Select A Storage Track Name

12.23 Select One Note Dialog

12.24 Select One Station Dialog

Chapter 13

Freight Car Forwarder (V2) Tutorial

The Freight Car Forwarder is a program designed to simulate freight car traffic on your model railroad. It does this by matching types of freight cars with industries. Specific types of freight cars are meant to carry specific types of commodities and specific industries produce or consume specific types of commodities.

Before you start using the Freight Car Forwarder system, you should carefully study Section Data files and Section Data File Formats of the reference section. These files describe the system layout (system file), the industries (industry file), the trains that will move the cars (trains file), and the cars themselves (the cars file). There are some additional files, including an owner's file and a car types file, as well as a file for statistics. All of these files are plain text files–you will need a plain text file editor (such as Notepad under MS-Windows or gedit under many versions of Linux [1]) to create and generally edit these files. The only files that are ever modified by the Freight Car Forwarder system are the cars and the statistics files. You should *not* edit the statistics file–this file is automatically generated by the Freight Car Forwarder system. While it is possible to use tools available as part of the Freight Car Forwarder system to edit cars in the cars file, it is probably best to use a regular text editor to add, modify, or delete cars in a wholesale manor. All of the other files are treated as "constant data" by the Freight Car Forwarder system, which will load the data into memory and not modify that data.

13.1 Loading System Data

The Freight Car Forwarder starts loading data by opening and reading the system file, using either the file menu's `Open...` item or open file button on the toolbar. This file contains the path names of the other files, which are assumed to be relative to the directory (folder) that contains the system file. All of the system data is loaded into a large data structure, which is then used by the program to simulate car movements.

13.2 Assigning Cars

In order to move cars, the cars need to be *assigned*, that is, they need to have a destination set, either to be loaded (if empty) or unloaded (if loaded). The Car Assignment procedure performs this task.

[1] It might be worthwhile to install a powerful general purpose text editor such as GNUEmacs for this purpose.

13.3 Running Trains

Once cars has been assigned, they need to be moved. Cars are moved on trains, and this is done with the run trains procedures. There are three of these procedures: Run All Trains in Operating Session, Run Boxmoves, and Run One Train at a time. The run trains procedures simulate the actual movement of cars and determines which trains will move which cars and in what order. From this simulation, a set of yard and switch lists can be generated and printed out for use during your operating session.

13.4 Printing Yard and Switch Lists

Once the trains have been run, yard and switch lists can be printed out, using the print yard lists menu.

13.5 Saving the updated car data

Once you have assigned cars, simulated train movements and created your Yard and Switch Lists, you should save the cars file. You should now be ready to run your trains in an operating session. If your session went off well, you will be ready to make car assignments for your next session. If there were problems during the operating session (such as bad ordered cars or late trains), you might have to make adjustments before running the car assignment process for the next session. You would use the car editor to make these adjustments. [2]

13.6 Generating Reports

Various reports can also be generated and printed using the reports menu.

13.7 Other activities

Other activities include adding, removing, and editing cars and displaying various state information, such as assigned and unassigned cars, car movement information, lists of trains, and lists of industries, stations, and divisions.

[2] It might be easier to use a text editor in the case of wholesale changes.

Chapter 14

Freight Car Forwarder (V2) Reference

The Freight Car Forwarder (V2) is a hybrid program, consisting of a Tcl/Tk GUI on top of a C++ class library. The GUI provides the user interface to the algorithms and data structures contained in the C++ class library. The program is based on Tim O'Connor's Freight Car Forwarder originally written in QBASIC. I first ported the program to a pure Tcl/Tk application. Then for better performance, I recoded the low-level guts (mostly heavy data indexing logic) to a C++ class library, using the STL to implement the various aggregate collections of objects, retaining Tcl/Tk for the GUI.

14.1 Command Line Usage

The name of the system file to load can be specified on the command line. See Section Opening and loading a system file. for more information.

14.2 Layout of the Main GUI

The main GUI window, shown below,

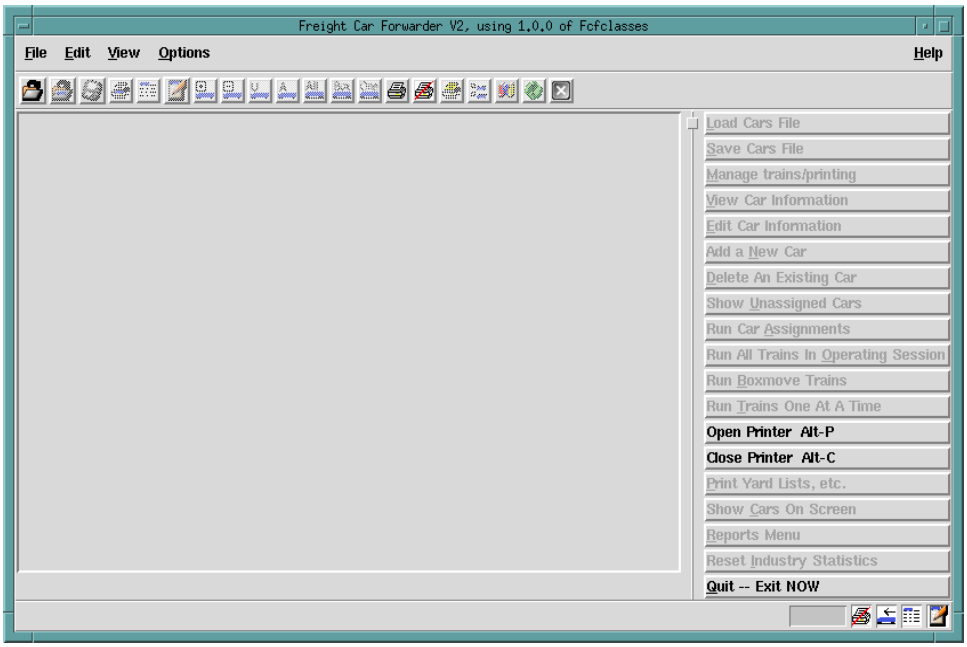


Figure 14.1 The main GUI screen of the Freight Car Forwarder (V2) Program

contains a menu bar, a toolbar,

Figure 14.2 The Toolbar of the Freight Car Forwarder (V2) Program

a text display area, and a button menu.

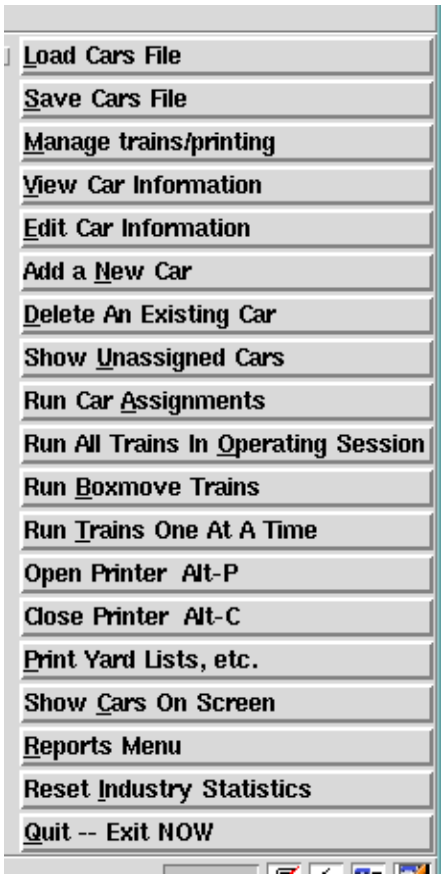


Figure 14.3 The Button Menu of the Freight Car Forwarder (V2) Program

There is also a work in progress message area, a general status area, a progress meter, and several indicators.

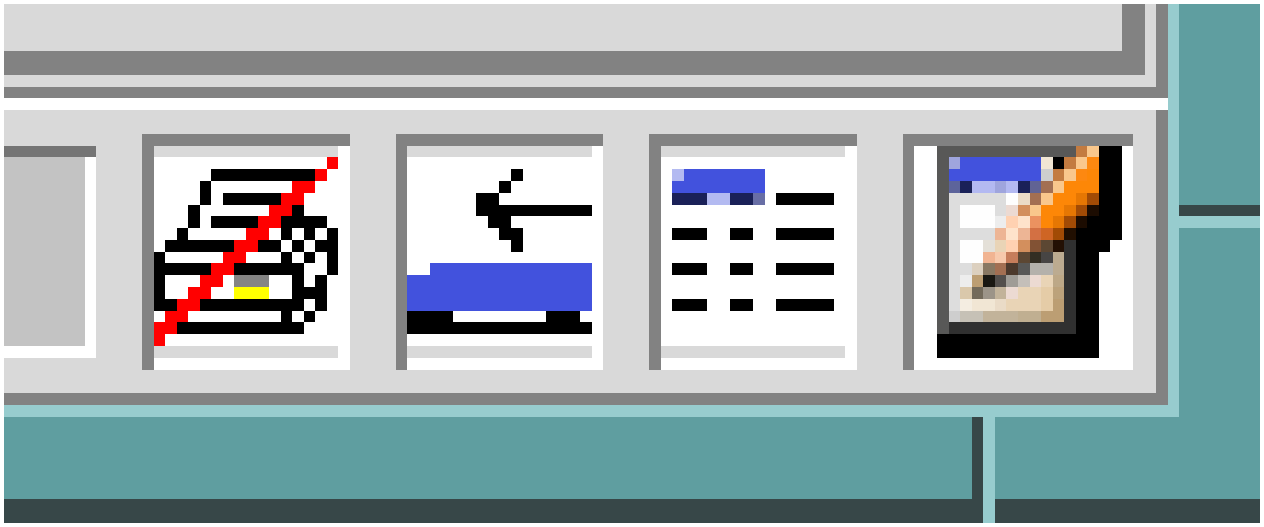


Figure 14.4 The Indicators of the Freight Car Forwarder (V2) Program

The main GUI also has three "slide out" frames, one for showing train status when trains are run, one for viewing a car's information, and one for editing a car's information. Each slide out has a corresponding indicator.

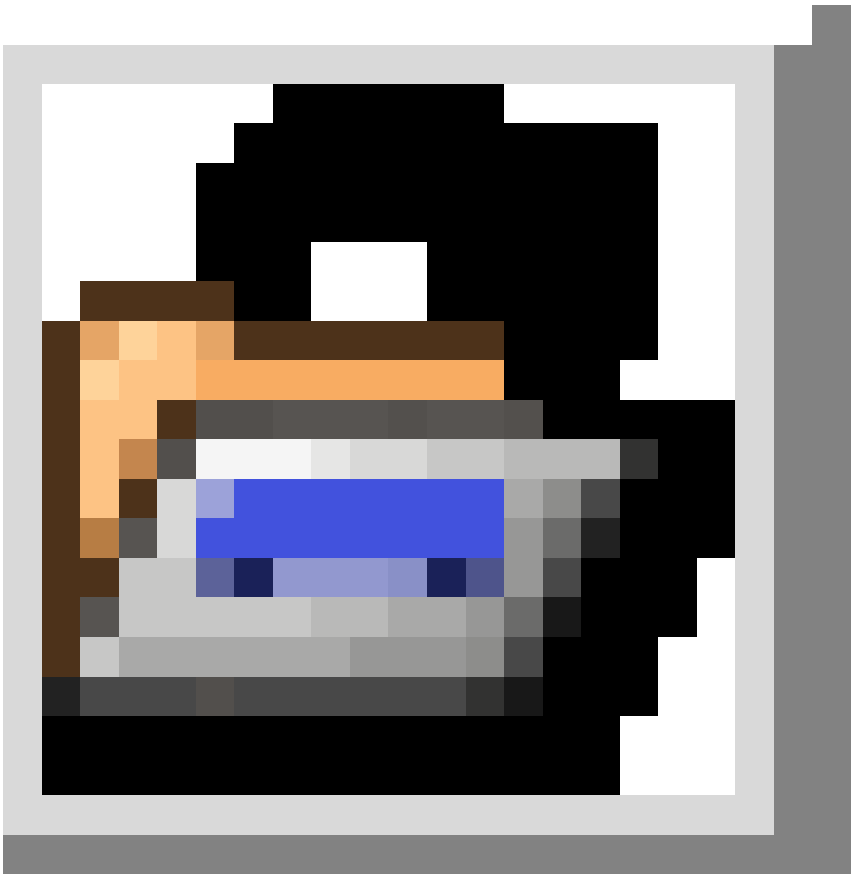
14.3 Opening and loading a system file.

The `File->Open...` menu button and the

toolbar button pop-up a file selection dialog to select a system file to load. Once this file is successfully loaded, the name of the file, the name of the system, the current session and shift number, plus a count of
divisions, stations, industries, cars, and trains is displayed in the main GUI's text area. Also all of the buttons are made active. The name of the system file can be specified on the command line and the named system file will be loaded when the program starts.

14.4 Loading and reloading the cars file.

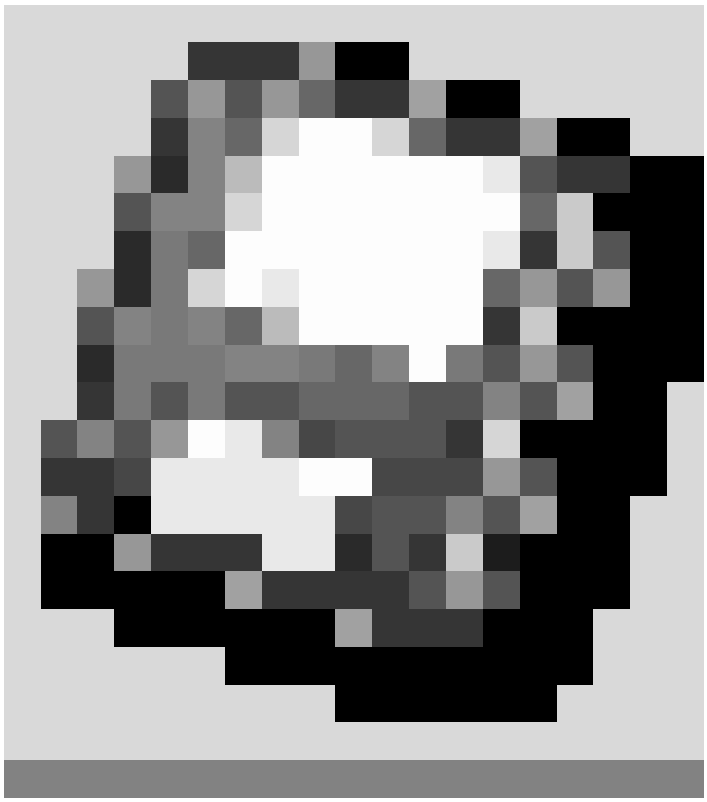
The `Load` Cars File menu button and the



toolbar button load (or reload) the cars file.

14.5 Saving the cars file.

The `Save Cars File` menu button and the



toolbar button save the cars and statistics files. This is something you need to do after you have simulated a session, by running the car assignment procedure and then run the trains in your session. This saves the state for the next time you run the Freight Car Forwarder.

14.6 Managing trains and printing

The `Manage` trains/printing menu button and the

toolbar button pop-up the train/printing management menu. This menu provides a set of functions relating to what trains are printed and can also print a dispatcher report and generate lists of various sorts of trains. The menu is shown below.

Figure 14.5 Train/Printing Management Menu.

14.6.1 Controlling Yard Lists

The `Control Yard Lists` menu item (y key) pops up a dialog, shown below, to control whether to print 0, 1, or 2 alphabetical lists and whether to print 0, 1, or 2 train lists.

Figure 14.6 Control Yard Lists Dialog

14.6.2 Enabling printing for all trains

The `Print All Trains` menu item (p key) turns on printing for all trains.

14.6.3 Disabling printing for all trains

The `Print No Trains` menu item (n key) turns off printing for all trains.

14.6.4 Printing a dispatcher report

The `Print Dispatcher Report` menu item (d key) enables the printing of a dispatcher report.

14.6.5 Listing local trains for this shift

The `List Locals This Shift` menu item (l key) lists all locals for this shift.

14.6.6 Listing manifests for this shift

The `List Manifests This Shift` menu item (m key) lists manifest freights for this shift.

14.6.7 Listing all trains for all shifts

The `List All Trains All Shifts` (? key) Lists all trains.

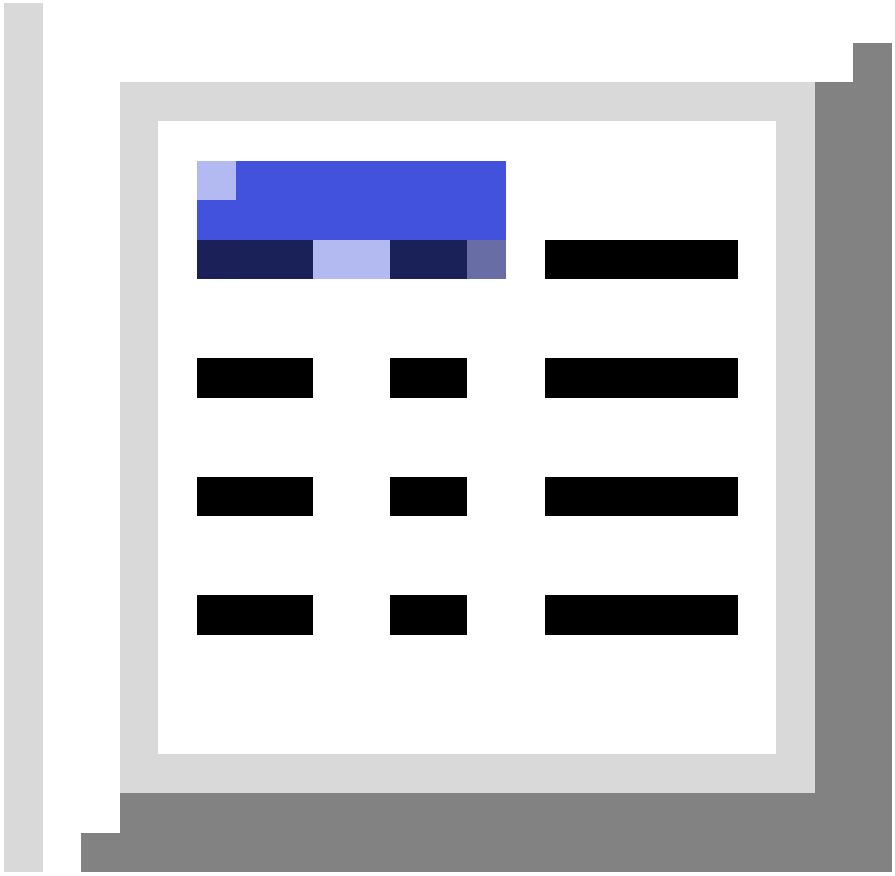
14.6.8 Managing one train

The `Manage One Train` menu item (1 key) pops up a dialog, shown below, to enable or disable printing of a single train, as well as setting the train's maximum length and setting which shift the train will be run. The train is selected with the "Select Train Dialog", described in Section Select A Train Dialog.

Figure 14.7 Train Management Dialog

14.7 Viewing a car's information

The `View Car Information` menu button and the



toolbar button display the information about a single car. The information is displayed on the view car "slide out", shown below. The car is selected with the "Search For Cars Dialog", described in Section Search For Cars Dialog.

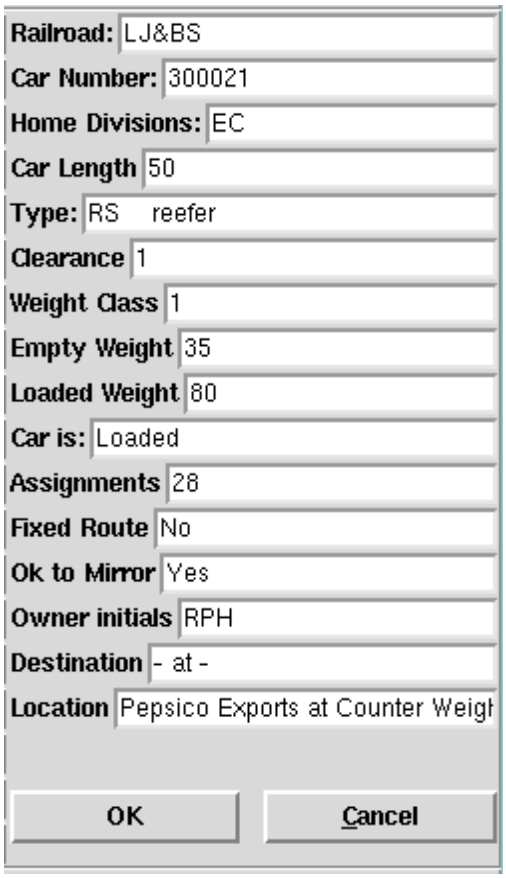


Figure 14.8 View Car Information Slideout

14.8 Editing a car's information

The `Edit Car Information` menu button and the

toolbar button display the information about a single car and allow for editing this information. The information is displayed on the edit car "slide out", shown below. The car is selected with the "Search For Cars Dialog", described in Section Search For Cars Dialog.

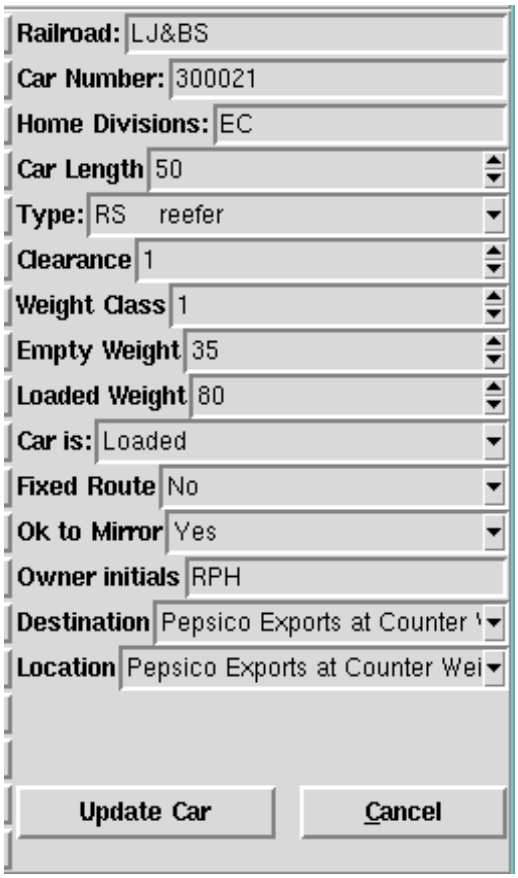
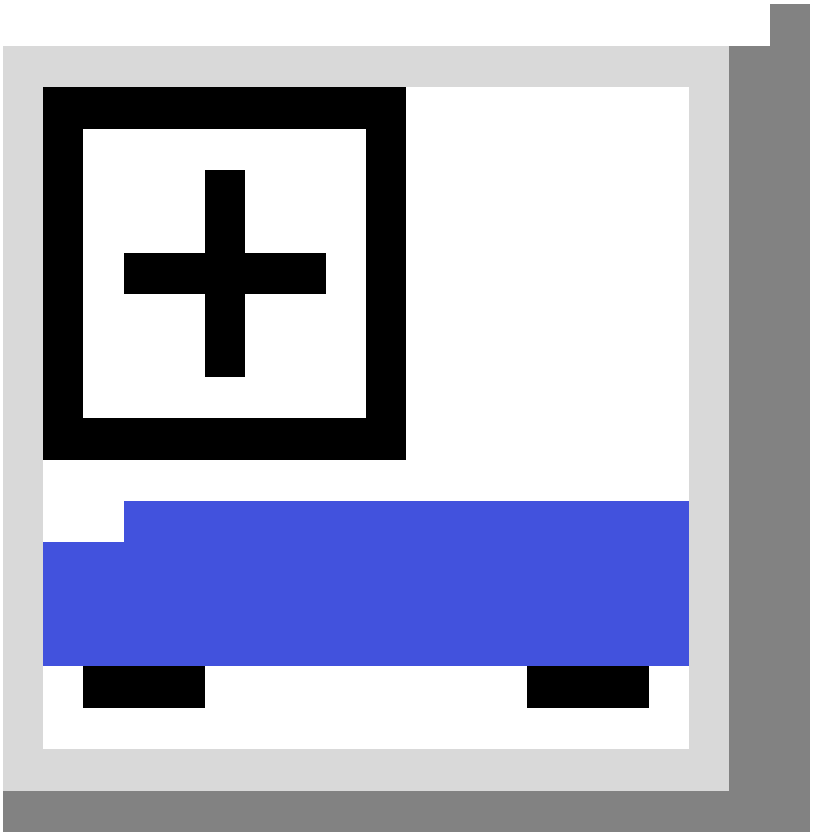


Figure 14.9 Edit Car Information Slideout

14.9 Adding a new car

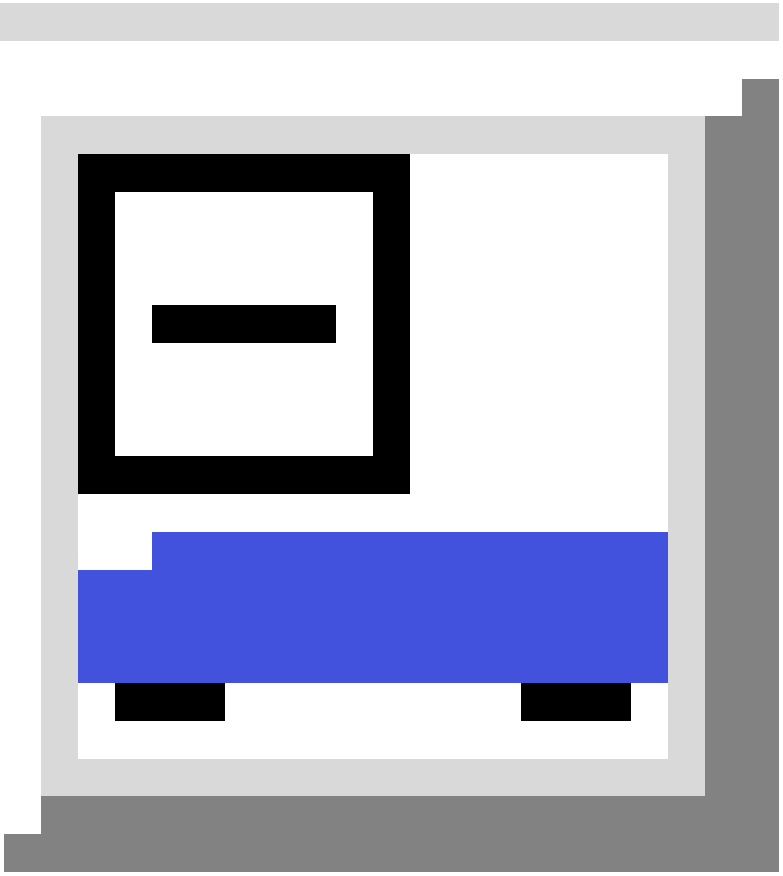
The `Add` a New Car menu button and the



toolbar button provide for adding a new car. The edit car "slide out", shown above, is displayed and the information about the new car can be filled in and the car added.

14.10 Deleting an existing car

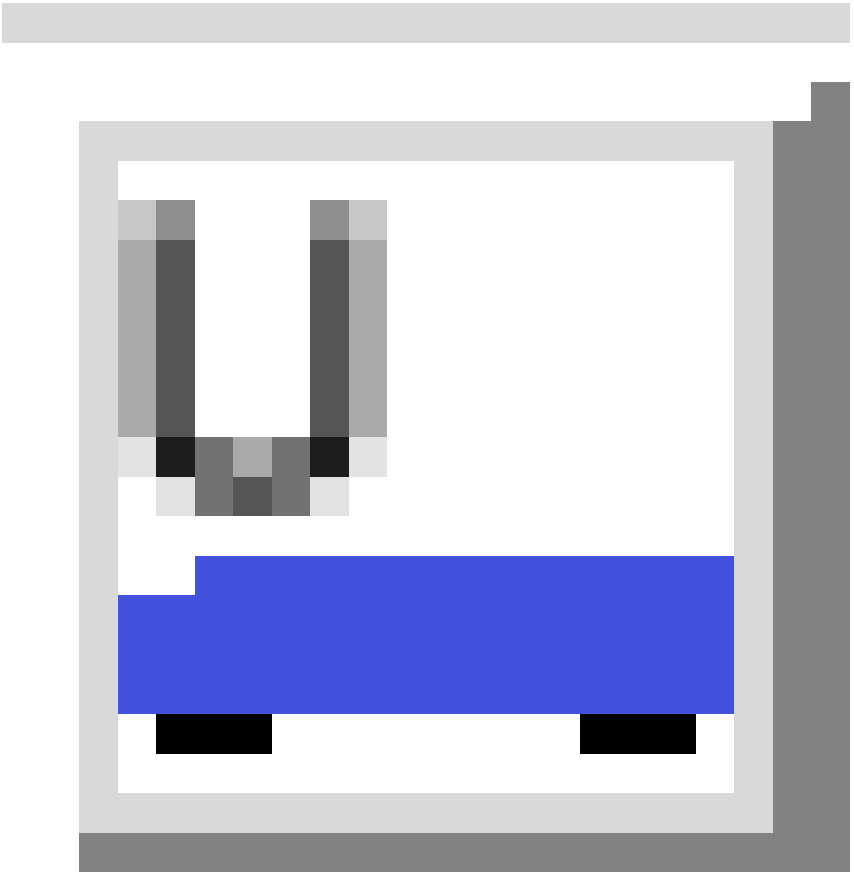
The `Delete An Existing Car` menu button and the



toolbar button provide for deleting an existing car. The car is selected with the "Search For Cars Dialog", described in Section Search For Cars Dialog and the car's information is displayed in the view car "slide out", shown above. Actual removal can then be confirmed.

14.11 Showing cars without assignments

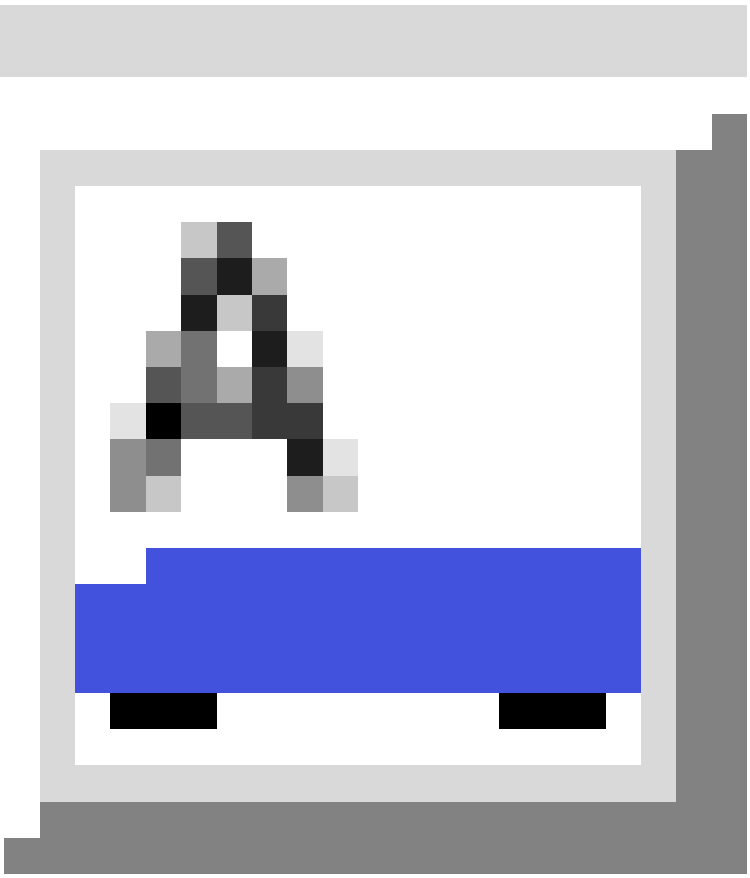
The `Show Unassigned Cars` menu button and the



toolbar button display unassigned cars in the text window.

14.12 Running the car assignment procedure

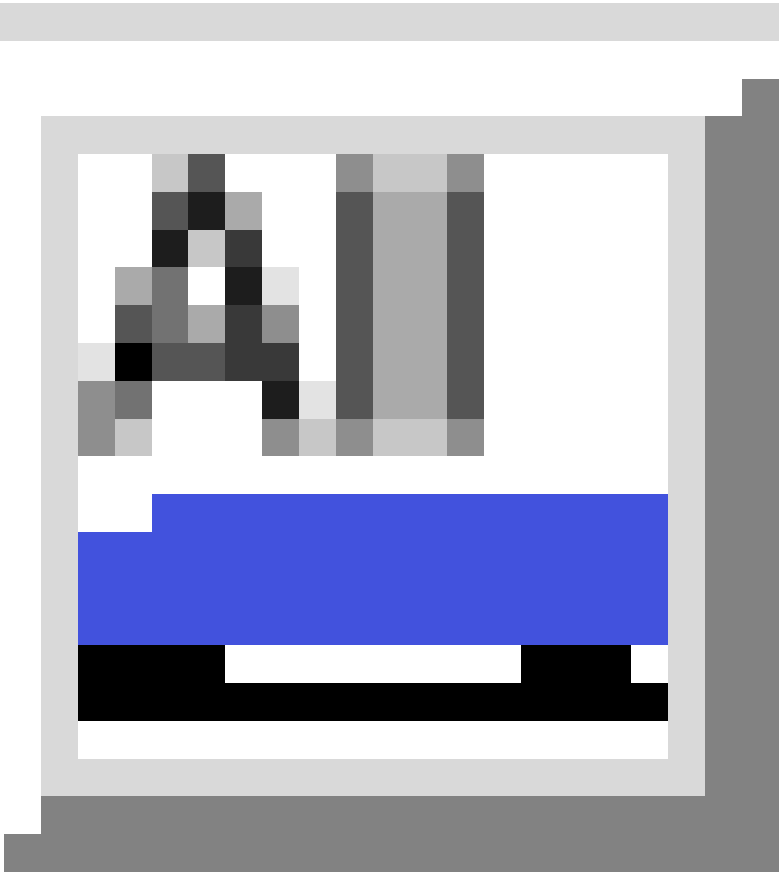
The `Run Car Assignments` menu button and the



toolbar button run the car assignment procedure. This procedure attempts to give as many unassigned cars assignments, that is possible destinations. Considerations taken into account are the type of car, whether it is loaded or not, industries with available trackage to accommodate the car, and so on. The list of cars is scanned twice and the progress of the procedure is displayed in the text area.

14.13 Running every train in the operating session

The `Run All Trains in Operating Session` menu button and the



toolbar button run all trains in the operating session, except the end of session box moves. Each train's progress is shown in the "Train Status Slideout", shown below.

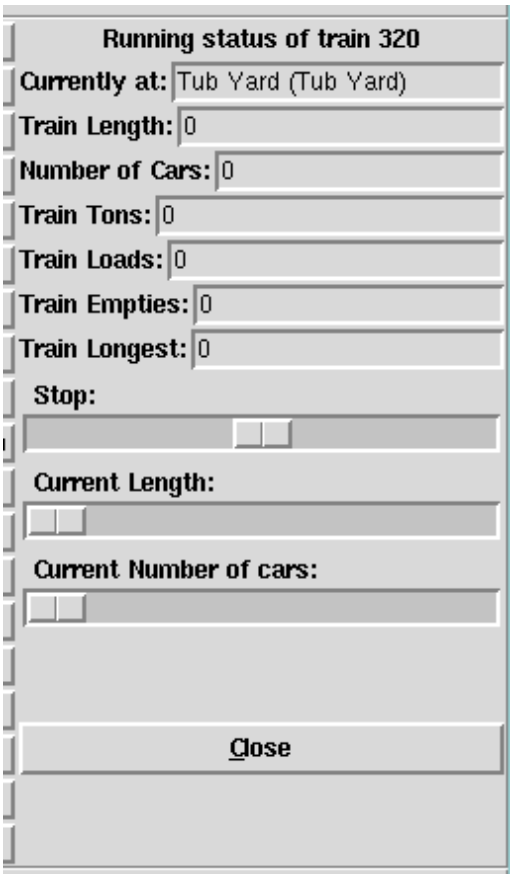
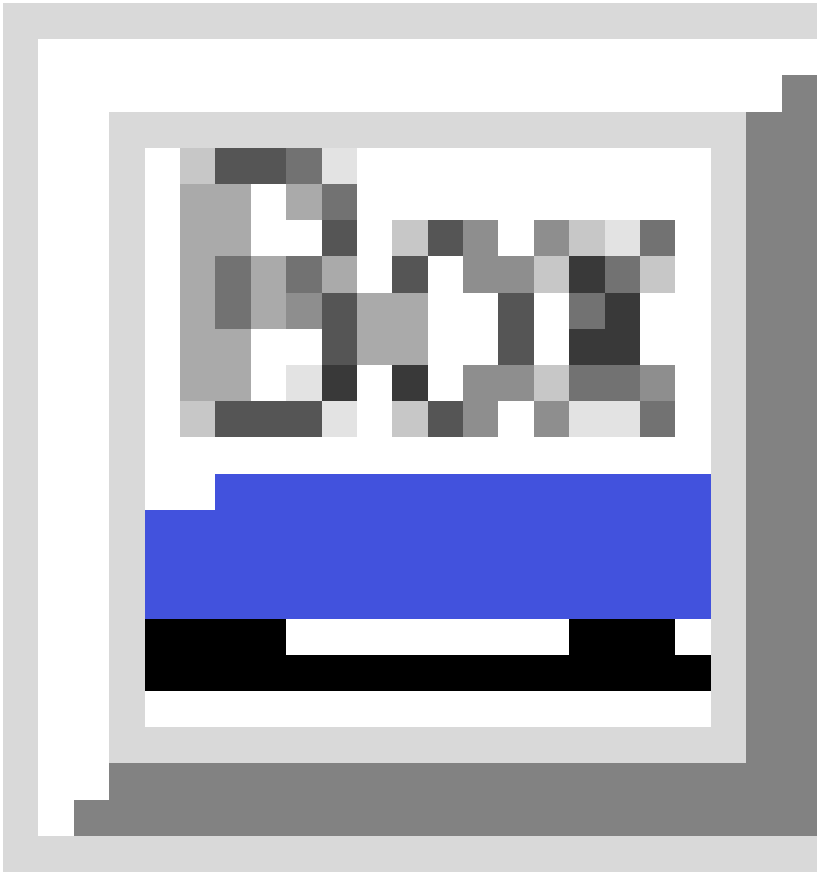


Figure 14.10 Train Status Slideout

14.14 Running the box move trains

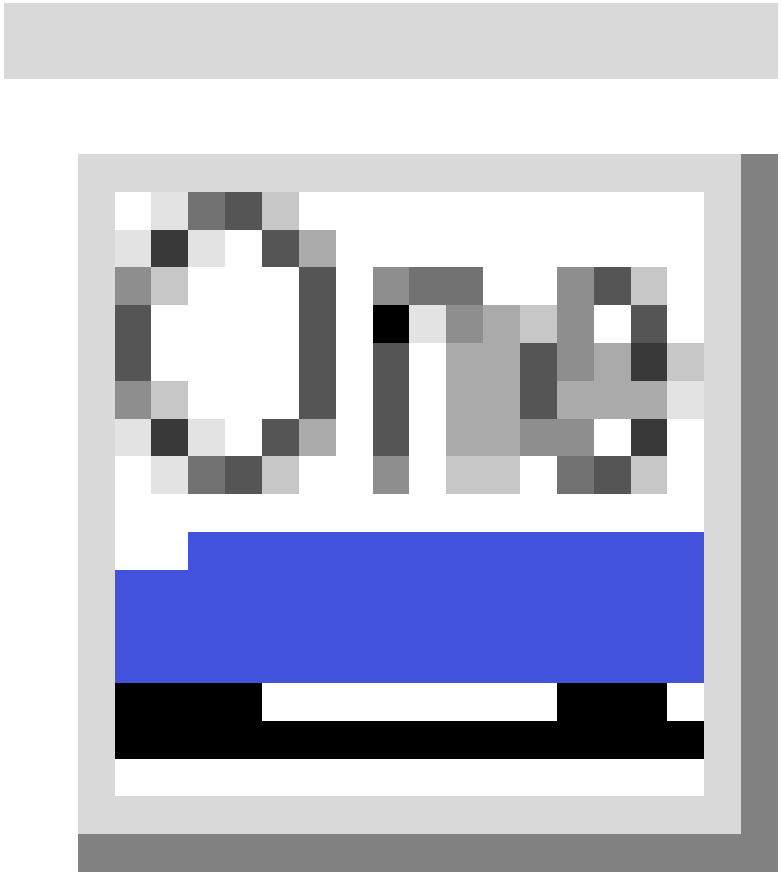
The `Run Boxmove Trains` menu button and the



toolbar button run all of the box move trains in the operating session.
Each train's progress is shown in the "Train Status Slideout", shown above.

14.15 Running a single train

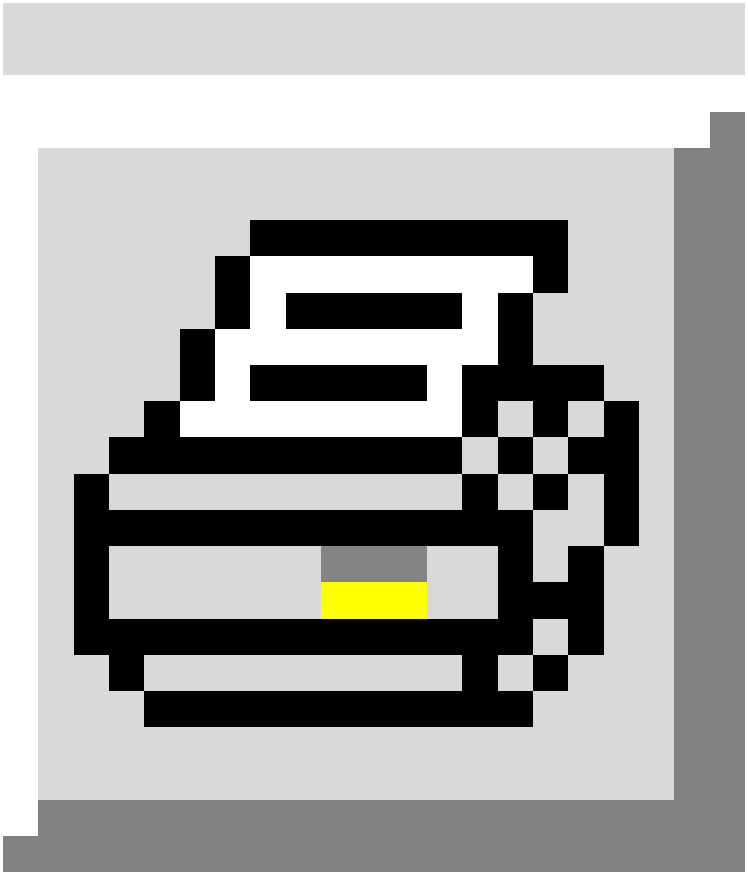
The `Run Trains One At A Time` menu button and the



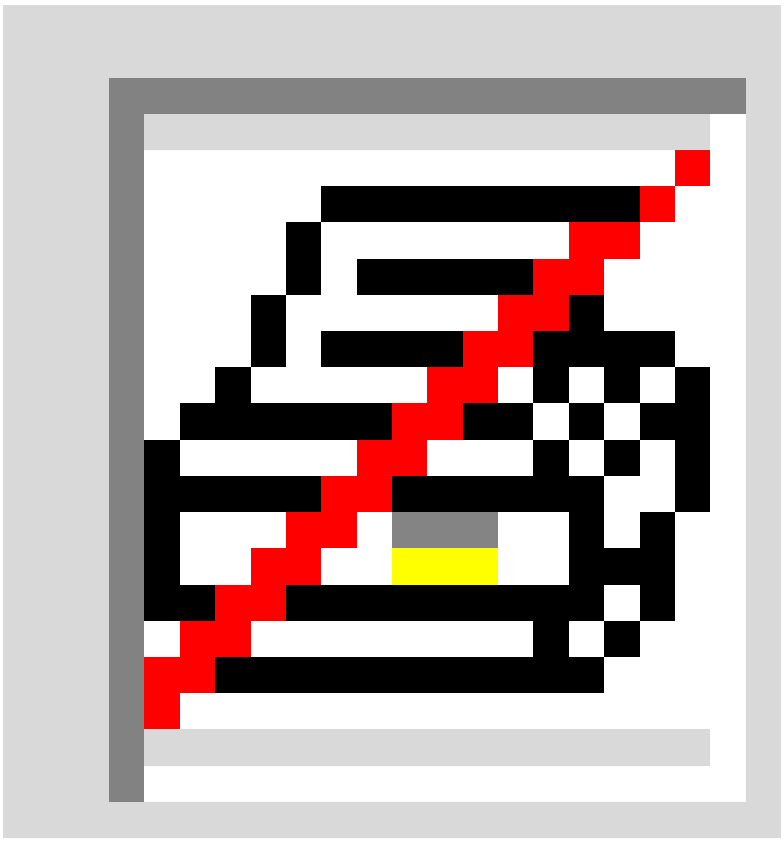
toolbar button run a single train, selected with the "Select Train Dialog", described in Section Select A Train Dialog. The train's progress is shown in the "Train Status Slideout", shown above.

14.16 Opening a Printer

The `Open Printer` menu button and the



toolbar button open the printer output file, using the "Open Printer Dialog", shown below. The status of the printer output, open or closed, is shown with the printer status indication.



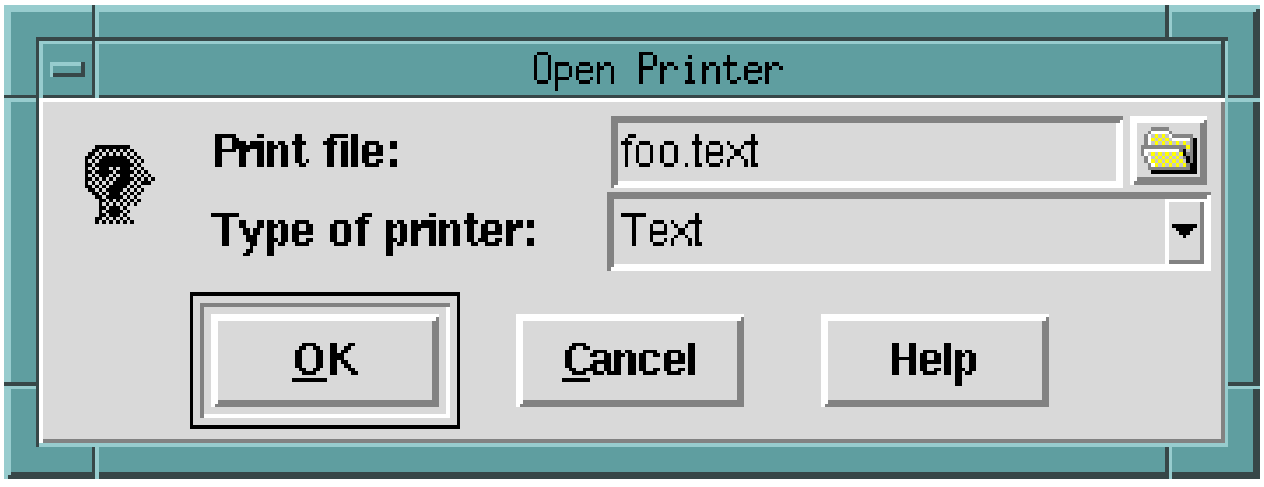


Figure 14.11 Open Printer Dialog

14.17 Closing the printer

The `Close Printer` menu button and the

toolbar button close the printer.The status of the printer output, open or closed, is shown with the printer status indication.

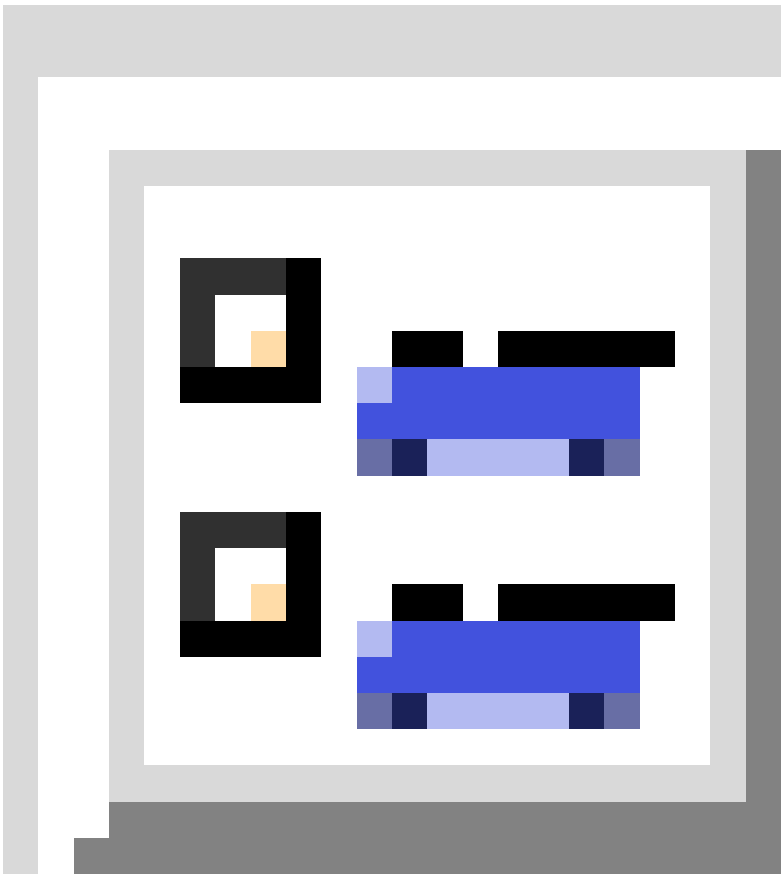
14.18 Printing yard and switch lists

The `Print Yard Lists`, etc. menu button and the

toolbar button print the yard and switch lists.

14.19 Showing cars on the screen

The `Show Cars On Screen` menu button and the



toolbar button pops up a menu, shown below, of classes of cars to show.

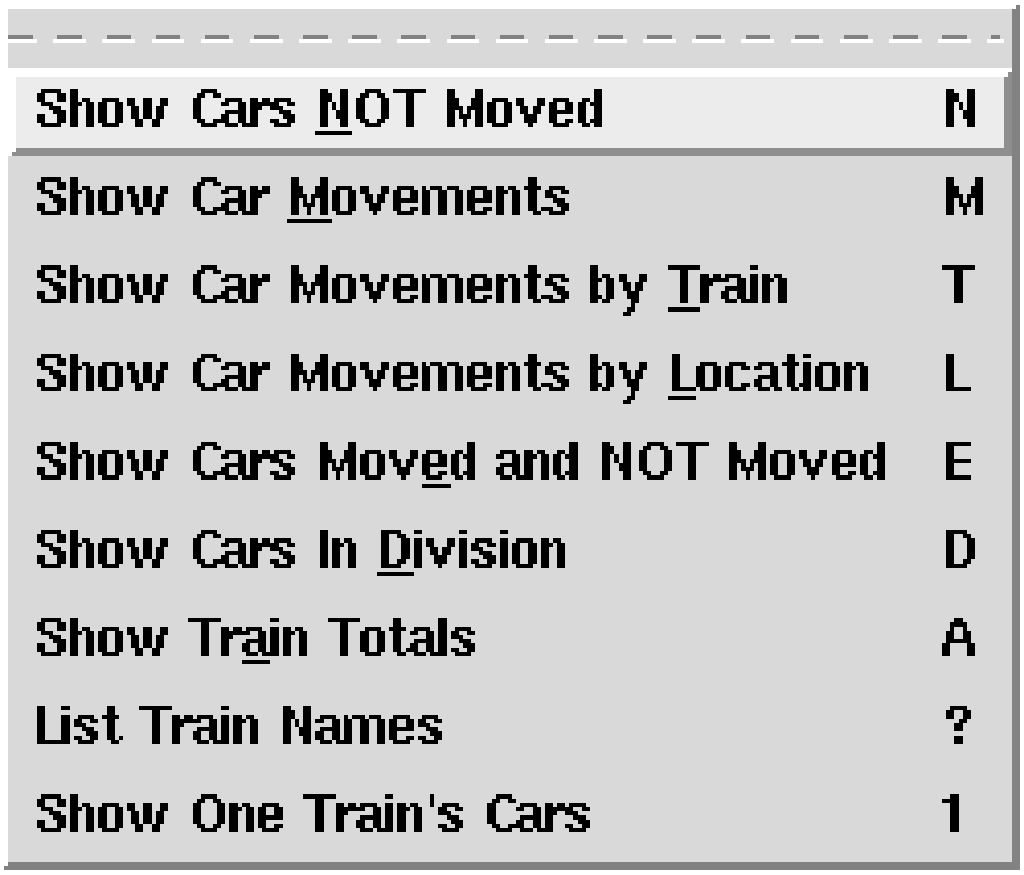
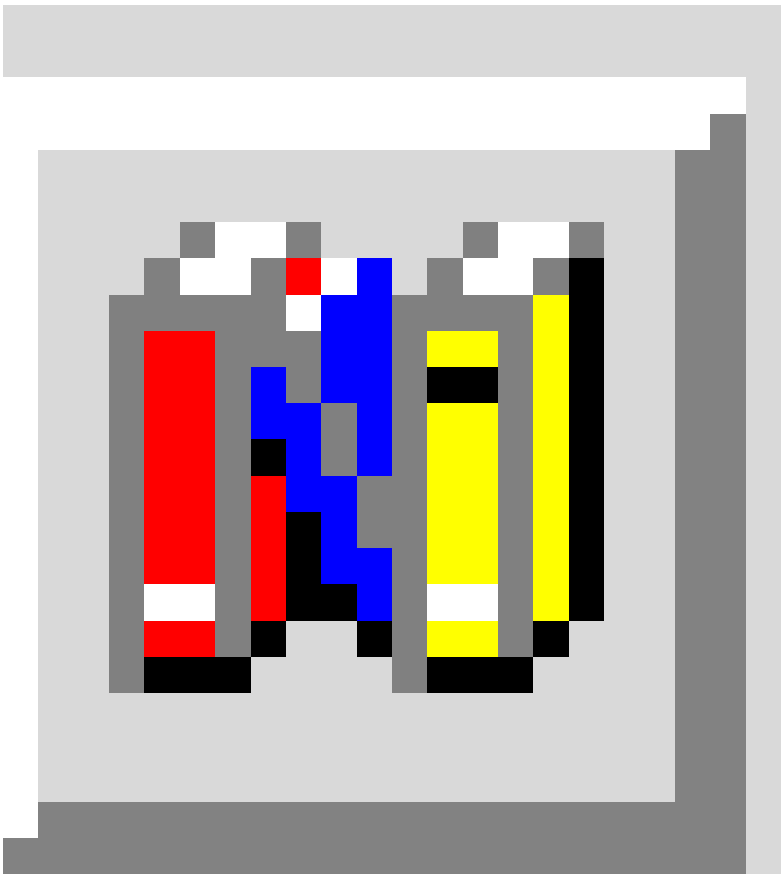


Figure 14.12 Show Cars Menu

14.20 Printing Reports

The `Reports Menu` menu button and the

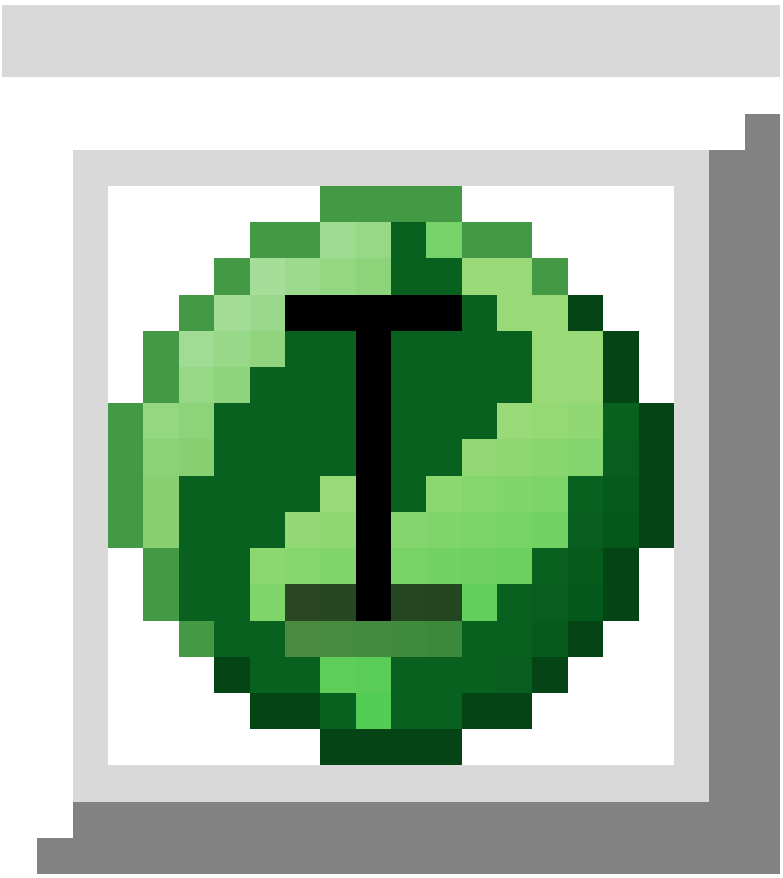


toolbar button pops up a menu, shown below, of possible reports.

Figure 14.13 Reports Menu

14.21 Resetting Industry Statistics

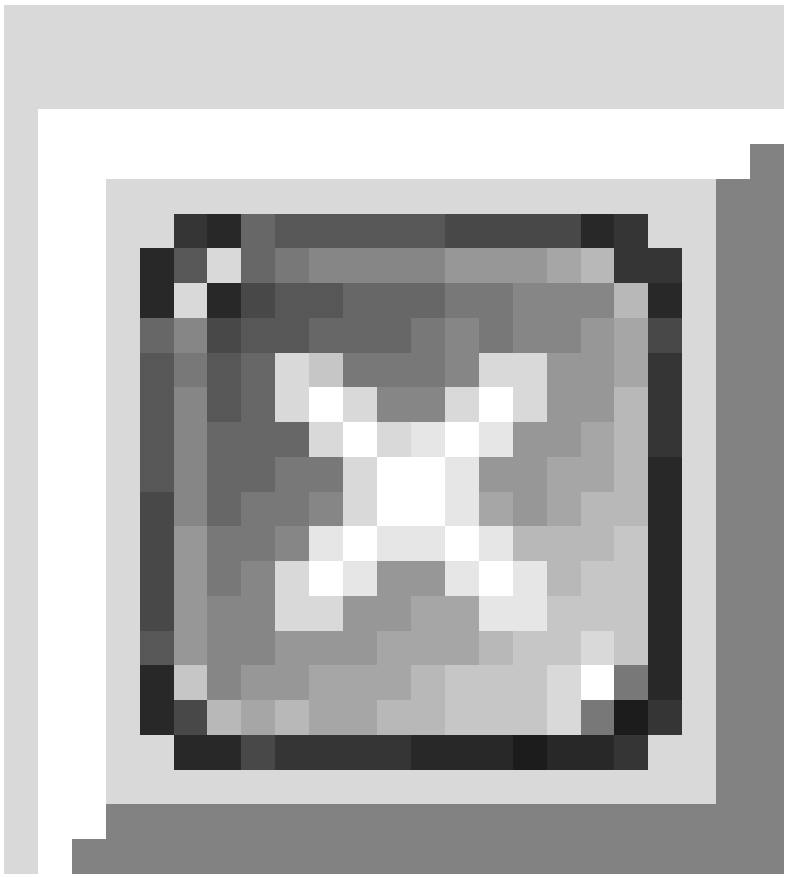
The `Reset Industry Statistics` menu button and the



toolbar button resets the industry statistics.

14.22 Quiting the application

The `Quit – Exit NOW` menu button and the



toolbar button exit the program. A confirmation dialog is popped up.

14.23 General Dialogs

14.23.1 Control Yard Lists Dialog

14.23.2 Enter Owner Initials Dialog

14.23.3 Select A Train Dialog

The Select a Train Dialog is used to select a train (to manage, run, or print). The `Filter` button uses the Train Name Pattern to match against train names to select a subset of trains to select from and can contain these special sequences:

- $*$ Matches any sequence of zero or more characters in the train name.
- ? Matches any single character in the train name.
- [chars] Matches any character in the set given by chars.

- If a sequence of the form `x-y` appears in chars, then any character between `x` and `y`, inclusive, will match. Characters are matched in a case insensitive way.

- `\x` matches the single character `x`. This provides a way of avoiding the special interpretation of the characters `*?[]\` in the pattern.

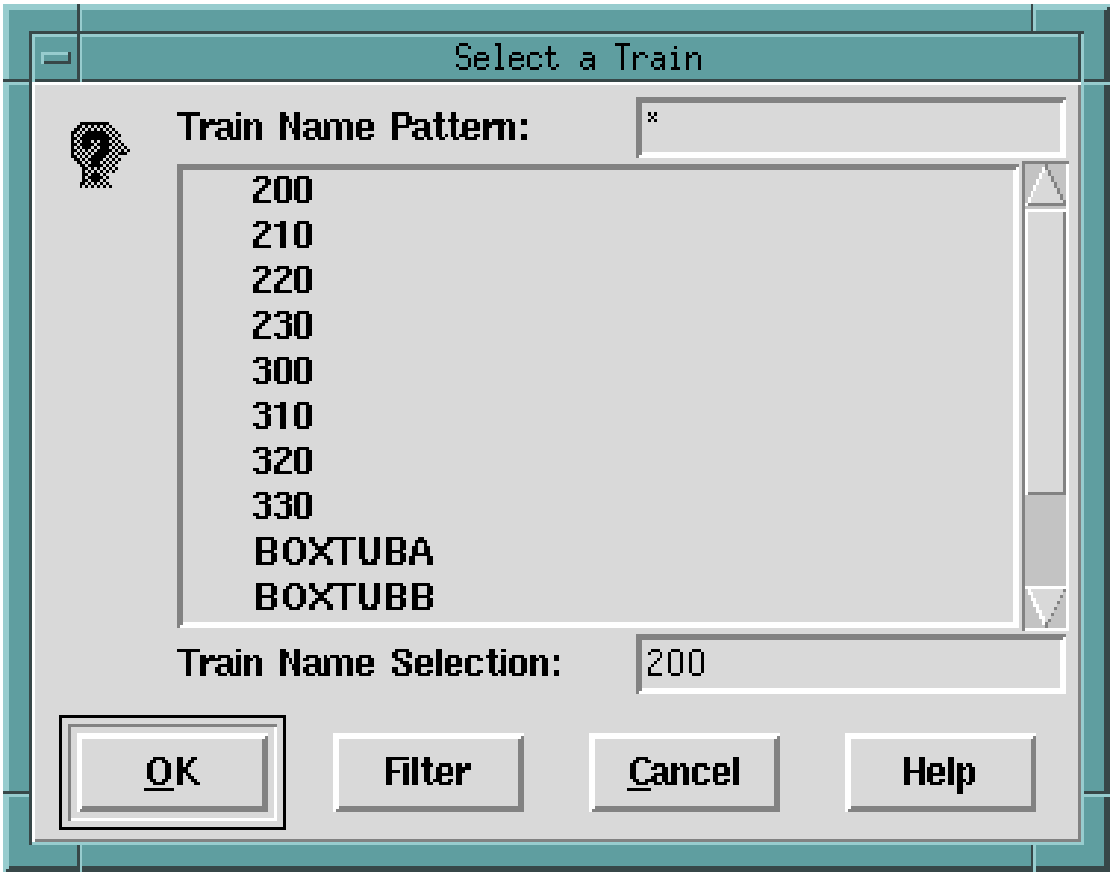


Figure 14.14 Select A Train Dialog

14.23.4 Manage One Train Dialog

14.23.5 Open Printer Dialog

14.23.6 Search For Cars Dialog

The Search For Cars Dialog is used to select a car (to view, edit, or delete). The `Filter` button selects a subset of cars based on the trailing car number digits.

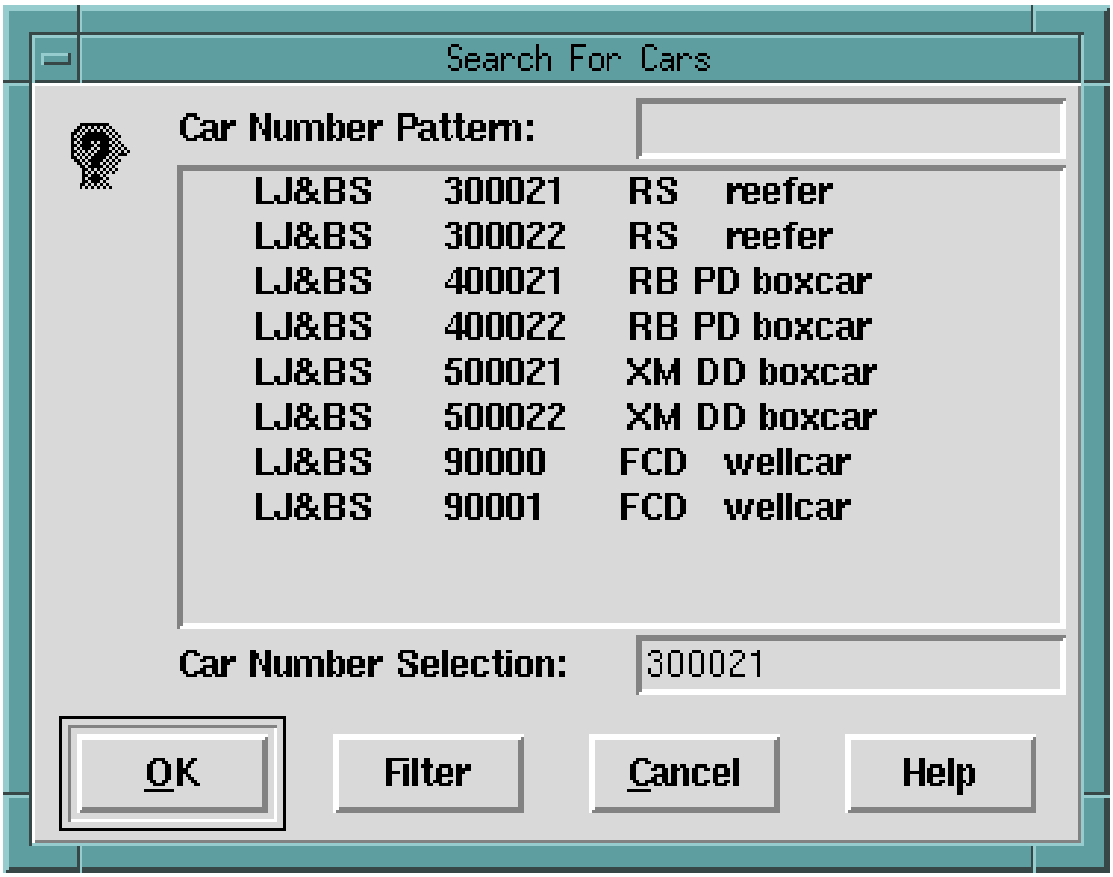


Figure 14.15 Search For Cars Dialog

14.23.7 Select A Division Dialog

14.23.8 Select An Industry Dialog

14.23.9 Select A Station Dialog

14.23.10 Select Car Type

14.24 Data files

The Freight Car Forwarder uses a collection of eight data files:

1. `System File` This is the **master** file. It contains the (relative) paths to the remaining seven files, along with the name of the railroad system, its divisions, and its stations.
2. `Industry File` This file holds the description of the industries, both on-line, which are actually modeled on the layout and off-line, which are imaginary industries not actually on the layout, but might be modeled as implied by staging yards or by interchange with other layouts or imaginary off-line railroads.

3. `Trains File` This file holds the description of the trains used to actually move the cars about the layout.
4. `Orders File` This file contains standing train orders and is only used to add additional information to the printouts given to train operators.
5. `Owners File` This file contains a mapping between owner initials and owner names. Used with various generated reports.
6. `Car Types File` This file contains a mapping between car type codes and full names and descriptions of car types.
7. `Cars File` This is the file containing information about all of the rolling stock on or off the layout.
8. `Statistics File` This is the statistics file. It is generated by the program and contains statistical information about car and industry utilization.

14.24.1 Data File Formats

Some general notes:

A comment it indicated by an apostrophe. All characters from the apostrophe to the end of the line are discarded when read. The files generally contain lines of comma separated fields, a format designed for BASIC read statements–the original program that this program is based on was written in a version of BASIC and uses the same file format.

14.24.1.1 System File

The first line of the system file is the name of the railroad system. This line is used in various banners and report headings.

The second line should be a blank line.

Then come the names of the remaining seven data files, one per line, in this order: `Industry File`, `Trains File`, `Orders File`, `Owners File`, `Car Types File`, `Cars File`, and finally `Statistics File`.

After the file names comes the division list. This starts with a count of the maximum number of divisions:
`Divisions = Number`

where Number is a positive non zero integer.

This is followed by division specifications, which is a list of 5 values separated by commas:
`Number,Symbol,Home,Area,Name`

Where Number is the index of the division (between 1 and the max number of divisions, inclusive), Symbol is an alphanumeric character (a-z, 0-9, A-Z), Home is the number of the home yard for this division (must be a yard specified in the `Industry File`), area is an Area symbol, and Name is the name of the division.

A line containing a -1 terminates the list of divisions.

Then comes the stations (cities), starting with a line defining the maximum number of stations:
`Stations = Number`

where Number is a positive non zero integer.

This is followed by station specifications, which is a list of 4 values separated by commas:
`Number,Name,Division,Comment`

Where Number is the index of the station (between 1 and the max number of stations, inclusive), Name is the name of the city, Division is the division index, and Comment is commentary about the station. City/Station number one is used for the workbench.

A line containing a -1 terminates the list of stations.

14.24.1.2 Industry File

The industry file contains industries and yards. The file starts with a line specifying the maximum number of industries:
`Industries = Number`

where Number is a positive non zero integer.

Followed by a line for each industry or yard. Industry number 0 is used for the repair yard, which is for cars not in service. Each industry's line contains these fields:
`ID,T,STA,NAME,TLEN,ALEN,P,R,H,MIR,C,W,DCL,MAX,LD,EM`

Where:

ID Numeric identifier.

T Types are **Y** for yard or **I** for industry or **O** for offline.

STA Station Identifier.

NAME User friendly place name.

TLEN Actual or virtual track length.

ALEN Assignable length.

P Priority for car assignments. If **YARD** or **STAGE**, **P** is n, the number of yard lists to print of type `A`, `P`, or `D`.

R Reloads cars **Y** for yes or **N** for no.

H Hazard class for outbound cargo.

MIR Mirror industry or 0 if none.

C Maximum clearance plate.

W Maximum weight class.

DCL Destination Control List of divisions. If **YARD** or **STAGE**, DCL can contain:

- **A** Alphabetical listing of cars in yard is permitted.
- **P** Pickup listing of cars in yard is permitted.
- **D** Dropoff listing of cars in yard is permitted.

MAX Maximum allowed car length.

LD Loaded car types accepted.

EM Empty car types accepted.

The industry listing is terminated by a line containing a -1.

14.24.1.3 Trains File

The trains file contains the trains used to move the cars. The file starts with a line specifying the maximum number of trains:

```
Trains = Number
```

where Number is a positive non zero integer.

Followed by a record for each train (a newline is acceptable alternative to a comma):

```
Number,Type,Shift,Done,Name,Maxcars, Divisions, Stops
     filler,Onduty,Print,Maxclear, Maxweight, Types,  Maxlen,
     Description
```

Where Number is the train number, Type is **M** for manifest; **B** for boxmove; **W** for wayfreight; or **P** for passenger, Shift is 1; 2; or 3, Done is **Y** for yes or **N** for no, Name is the train name, Maxcars is the maximum number of cars, Divisions is a set of division symbols or a wildcard (*),Stops is a space separated list of stations (Boxmove and Wayfrieghts) or industries (Manifests), filler is an unused slot (use 0), Onduty is the time on duty (the train's departure time) in the format HHMM, Print is **P** for print or **N** for noprint, Maxclear is the maximum clearance number, Maxweight is the maximum weight number, Types is a set of car types this train can carry, Maxlen is the maximum train length in feet, and Description is a textual description of the train.

The train listing is terminated by a line containing a -1.

14.24.1.4 Orders File

This file contains lines with pairs:

```
Name,Order
```

where Name is the name of a train and Order is a quoted string containing the order.

14.24.1.5 Owners File

This file starts with a count of owners and then lines with with triples:

```
Initials,Name,Comment
```

where Initials are the three letter initials of an owner, Name is the full name of the owner, and Comment is some descriptive text.

14.24.1.6 Car Types File

This is a file with exactly 91 records. Each record contains:

```
Car Type Code,Car Type Group,Description,pad,Comment
```

where Car Type Code is one of 91 printable characters, Car Type Group is a single character, Description is a 16 character brief description, pad is 0, and Comment is some descriptive text.

After the car types is the Car type groupings, which map groups of car types into groups using the second single character, with lines containing these fields:

```
Car Type Group,Description,Comment
```

where Car Type Group is a single character, Description is a 16 character brief description, and Comment is some descriptive text.

14.24.1.7 Cars File

The cars file starts with three numbers, one per line:

```
Total shifts
Current shift
Max car count
```

The first number is the total number of shifts, the second is the current shift number (1, 2, or 3), and the third number is the maximum number of cars in the file.

The remainder of the file is car records. This file must be kept in **alphabetical order!** Each record contains:

```
Type,Marks,Number,Home,CarLen,ClearPlate,CarWeight,EmptyWt,
        LoadLimit,Loaded,Mirror?,Fixed?,Owner,Done,Last,Moves,Loc,
        Dest,NTrips,NAssigns
```

Where Type is from car types file, Marks is the railroad reporting marks (9 characters max), Number is the car number (8 characters max), Home is car home division (from system file), CarLen is extreme car length, ClearPlate is the clearance plate (from plate file), CarWeight is car weight class (from weight file), EmptyWt is light weight in tons, LoadLimit is load limit in tons, Loaded is **L** for loaded or **E** for empty, Mirror? is ok to mirror **Y** for yes or **N** for no, Fixed? is fixed route **Y** for yes or **N** for no, Owner is car owner's 3 character initials (from owners file), Done is car is done moving for this session **Y** for yes or **N** for no, Last is last train to handle the car from trains file,Moves is actual movements this session,Loc is car's present location from industry file, Dest car's destination from industry file, NTrips is number of car trips, and NAssigns is number of car assignments.

14.24.1.8 Statistics File

The statistics is a file generated as an output and should not be hand edited. This file has two formats, V1 and V2. V1 is the original format used by the original BASIC program. V2 is an improved version that avoids getting the fields jammed together due to numerical overflow (result numbers too large for the field sizes).

The first line of either format contains the statistics period number. If in the new format (V2), this number is followed by a comma.

The rest of file file contains lines of four numbers, either space separated (V1) or comma separated (V2): industry index, car count, car length, and statistics length.

14.24.1.9 Other data files

There are some additional data files, which are not actually loaded into the system. These are the plate, weight, and hazard files. These are just informational files that are used to map clearance plate, weight class, and hazard levels of cars.

Chapter 15

Resistor Program Reference

The Resistor Calculator program aids in calculating dropping resistors for LEDs and low-voltage lamps commonly used on model railroads. It implements Ohm's Law, shown in the equations below to perform the calculation and then finds the nearest stock value and also displays the color bands for typical carbon resistors.

$$R_{drop} = \frac{V_{drop}}{I} \tag{15.1}$$

$$V_{drop} = V_{supply} - V_{load} \tag{15.2}$$

The calculator takes three input values, the supply voltage (V_{supply}), the voltage across the load (V_{load}) (LED or lamp) and the load current (I) the LED or lamp operates at. These values are entered along with the units they are in. Then the calculate button is pushed and the results are displayed. The results can also be saved to a text file, which can be printed or otherwise referred to later.

The main GUI screen of the Resistor Calculator program is shown here:

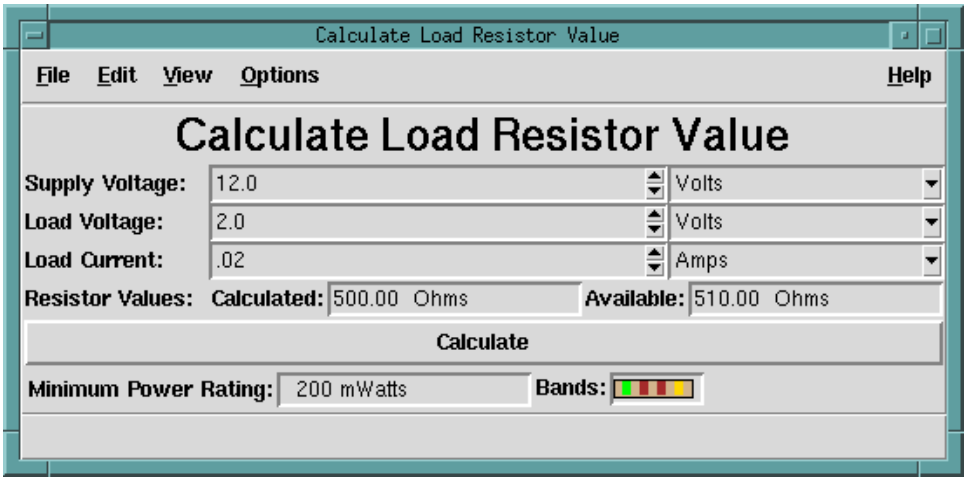


Figure 15.1 The main GUI screen of the Resistor Calculator program

Chapter 16

LocoPull Program Reference

16.1 Basis and Mathematics

This program is based on the information posted by Mark U. on the Yahoo XTrkCad list at the URL http://groups.yahoo.com/group/XTrkCad/message/4983 and the information supplied by rtroop on the TrainBoard forum in post no. 9 at the URL http://www.trainboard.com/grapevine/showthread.php?t=114497. This is a standalone program that incorporates the formulas presented in Mark U's spreadsheet and explained in rtroop's message. The formulas are as follows:

$$E_{unit} = W_{unit}A \tag{16.1}$$

$$E = E_{unit}N \tag{16.2}$$

$$R_{ave} = W_{ave}F \tag{16.3}$$

$$C_{0\ Grade} = \left\lfloor \frac{E}{R_{ave}} \right\rfloor \tag{16.4}$$

$$R_{grade} = W_{ave}G \tag{16.5}$$

$$R_{net\ at\ grade} = R_{ave} + R_{grade} \tag{16.6}$$

$$R_{unit\ at\ grade} = W_{unit}G \tag{16.7}$$

$$D = \frac{5730}{\frac{rS}{12}} \tag{16.8}$$

$$C_{grade\ and\ curve} = \left\lfloor \frac{E - NW_{unit}(G + F + D)}{R_{ave} + W_{ave}(G + F_{per\ degree}D)} \right\rfloor \tag{16.9}$$

Where:

W_{unit} is the weight of each locomotive in ounces.

A is the adhesion factor generally 25%.

E_{unit} is the tractive effort per unit in ounces.

E is the net tractive effort in ounces.

N is the number of units.

F is the resistance factor of each car, typically 4% for N scale cars.

W_{ave} is the average weight per car, typically 1 ounce for N scale cars.

R_{ave} is the average rolling resistance of each car.

$C_{0\ Grade}$ is the capacity of the train on level, straight track.

G is the percent of grade.

R_{grade} is the added rolling resistance of each car due to grade.

$R_{net\ at\ grade}$ is the net rolling resistance of each car at grade.

r is the track radius in inches.

S is the scale factor (160 for N scale, 87 for H0 scale, etc.).

D is the degree of curvature.

$F_{per\ degree}$ is the resistance factor per degree of curvature, typically .04%.

$C_{grade\ and\ curve}$ is the capacity of the train on at grade on a curve.

16.2 The GUI

The main GUI screen of the LocoPull program is shown below. The GUI is broken down into sections:

- The Scale section. The scale is selected here.
- The Locomotive Information section. Information about the locomotives is entered here. The number of locomotives, how much they weigh each, and their adhesion factor. The tractive effort for each unit and the net tractive effort are computed and displayed here. It is assumed that all of the powered engines are the same, typically the same make and model, with the same weight and same adhesion factor.
- The Consist Information. Information about the cars, including their average weight and their average resistance factor are entered and the rolling rolling resistance is computed and displayed.
- The Zero-grade Capacity section. The maximum number of cars that can be pulled on a straight track on a level grade is computed and displayed here.
- The Grade Information section. The percent of grade is entered and the added rolling resistance per car at grade, the net rolling resistance, and the added resistance per unit are computed and displayed here.
- The Curve Information section. The radius of the curve in inches and the rolling resistance per degree of curve are entered and the degree of curvature is computed and displayed.
- The Capacity at Grade and Curve section. This is the maximum number of cars that can be pulled at the grade and curve specified.
- Calculate button. This button performs the calculation and updates all of the displayed values.

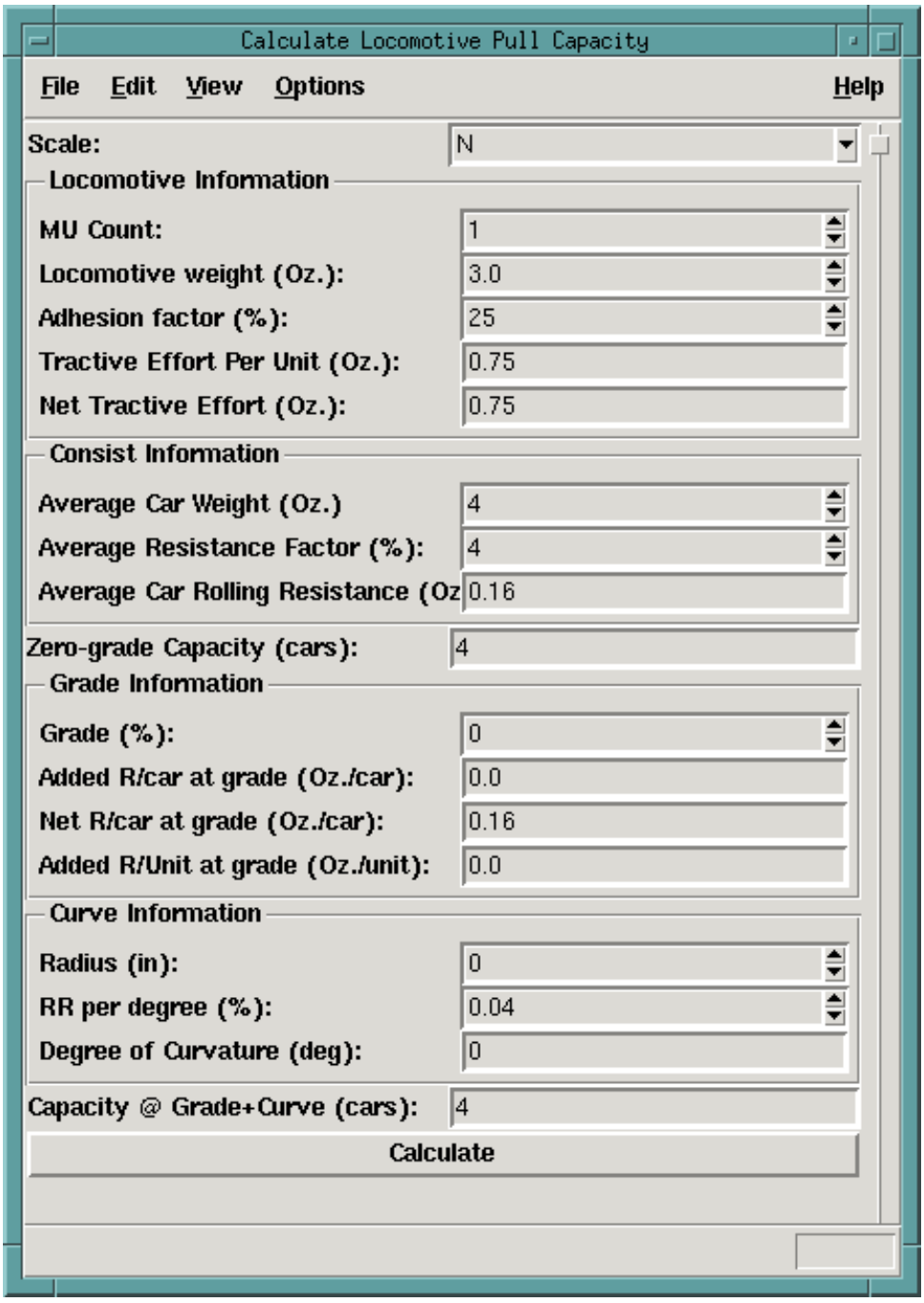


Figure 16.1 The main GUI screen of the LocoPull program

16.2.1 The Scale

The scale selection simply select the scale and is used to compute the degree of curvature.

16.2.2 Locomotive Information

This section of the GUI gathers information about the locomotives pulling the train. It is assumed that all of the locomotives have the same tractive effort, that is they are the same weight and have the same adhesion factor. This would generally be the case if the locomotives were the make and model. Three inputs are gathered in this section: the number of locomotives, the weight of each locomotive, and the adhesion factor of the locomotives. Two intermediate outputs are displayed here: the tractive effort of each locomotive and the net tractive effort of all of the locomotives together.

16.2.3 Consist Information

This section gathers two inputs and displays one intermediate result. The two inputs are the average weight of the cars and the average rolling resistance factor. The intermediate result is the average car rolling resistance.

16.2.4 Zero-grade capacity

This is simply the net tractive effort divided by the average car rolling resistance. The floor of the result is displayed as a whole number (since pulling a fraction of a car is not meaningful).

16.2.5 Grade information

One input is gathered and three intermediate results are displayed. The input is the percent of grade and the intermediate results displayed are the added rolling resistance at grade of each car, the net rolling resistance per car, and the added rolling resistance of each locomotive at grade.

16.2.6 Curve information

This section gathers two inputs and displays one intermediate result. The added inputs are the curve radius and the rolling resistance per degree of curvature and the intermediate result is the degree of curvature.

16.2.7 Capacity and Grade plus Curve

This is just the tractive effort less the tractive effort needed to pull the locomotives themselves divided by the combined rolling resistance of the average car: base rolling resistance plus the added rolling resistance due to grade, plus the added rolling resistance due to the curvature. The floor of the result is displayed as a whole number (since pulling a fraction of a car is not meaningful).

Chapter 17

Camera Programs Reference

AnyDistance and Closest compute the view angle in both real and scale units. It also computes the effective scale of the imaging plane, such as the size of a 35mm slide, which might be used as a transparency for model window panes or locomotive number boards.

Both programs work the same. The only difference is that Closest uses the closest effective focus of the lens and Any↩ Distance uses a user specified focus distance. Given the input parameters, the distance, the lens, the scale, and the film size, a diagram is displayed with the dimensions of the view. This diagram can be printed using the `Print...` menu item under the `File` menu.

New lenses can be entered with the `New` menu item under the `File` menu. The `Open...` and `Save..` menu items can load and save the set of available lenses.

Both programs solve the equation below and display a diagram illustrating the solution. AnyDistance uses a user entered value for D and Closest uses the closest focusing distance for the selected lens.

$$W_{view} = (DS)2\tan(\frac{\theta}{2}) \tag{17.1}$$

Where:

$$\begin{aligned} W_{view} &= \text{The scale view width.} \\ D &= \text{The distance between the scene and the camera lens.} \\ S &= \text{The model scale factor.} \\ \text{and} & \\ \theta &= \text{The lens view angle.} \end{aligned}$$

The main GUI screen of the AnyDistance program is shown below. The Closest program is much the same, except that the distance parameter is omitted.

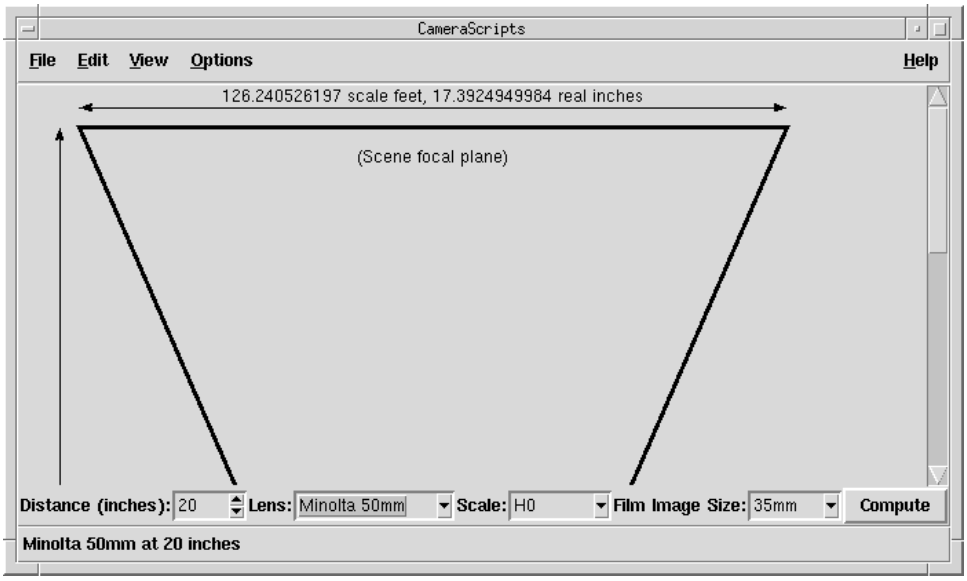


Figure 17.1 The main GUI screen of the AnyDistance program

Chapter 18

Dispatcher Tutorial

18.1 A "Simple Mode" CTC Panel

This tutorial will go through the steps of creating a simple CTC panel for a passing siding. First, after starting up the Dispatcher program, we will click on the `New CTC Window` toolbar button and get a `New CTCPanel` dialog box, as shown below. See Section Creating a new CTC Panel for more information.

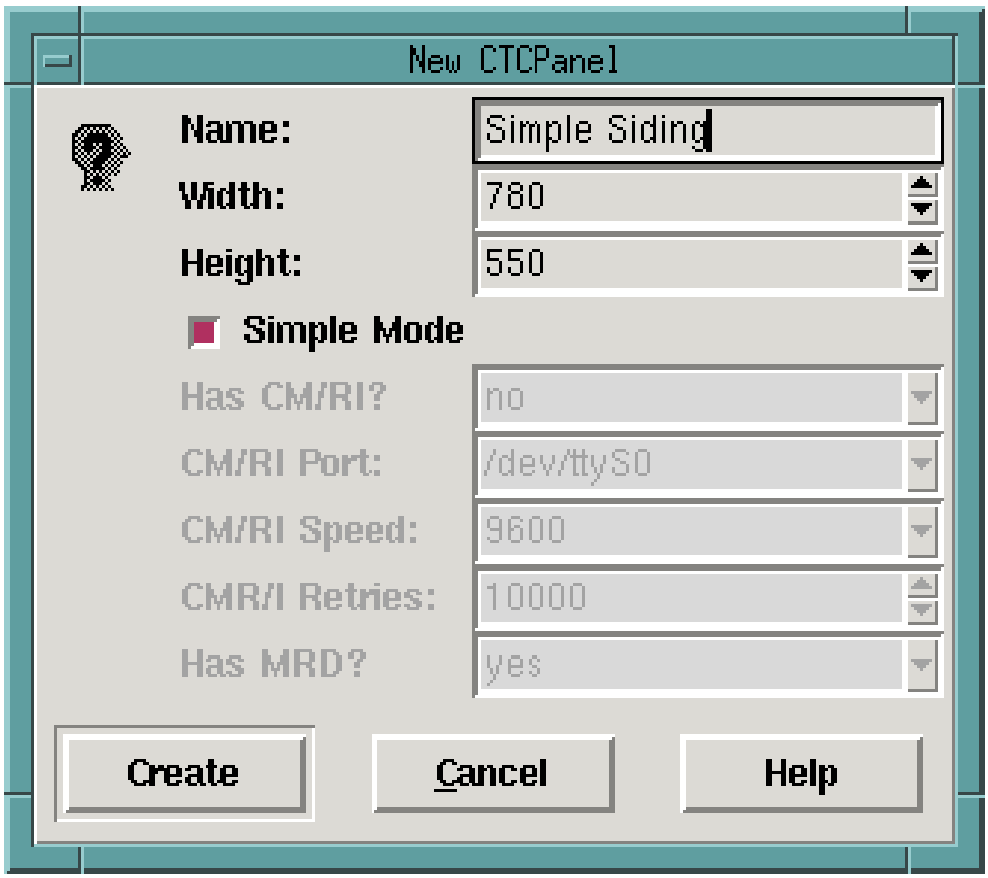


Figure 18.1 Creating a new Simple Mode CTC Panel

We fill in a name and select the `Simple Mode` check button. Clicking on `Create` gives us the blank panel shown below.

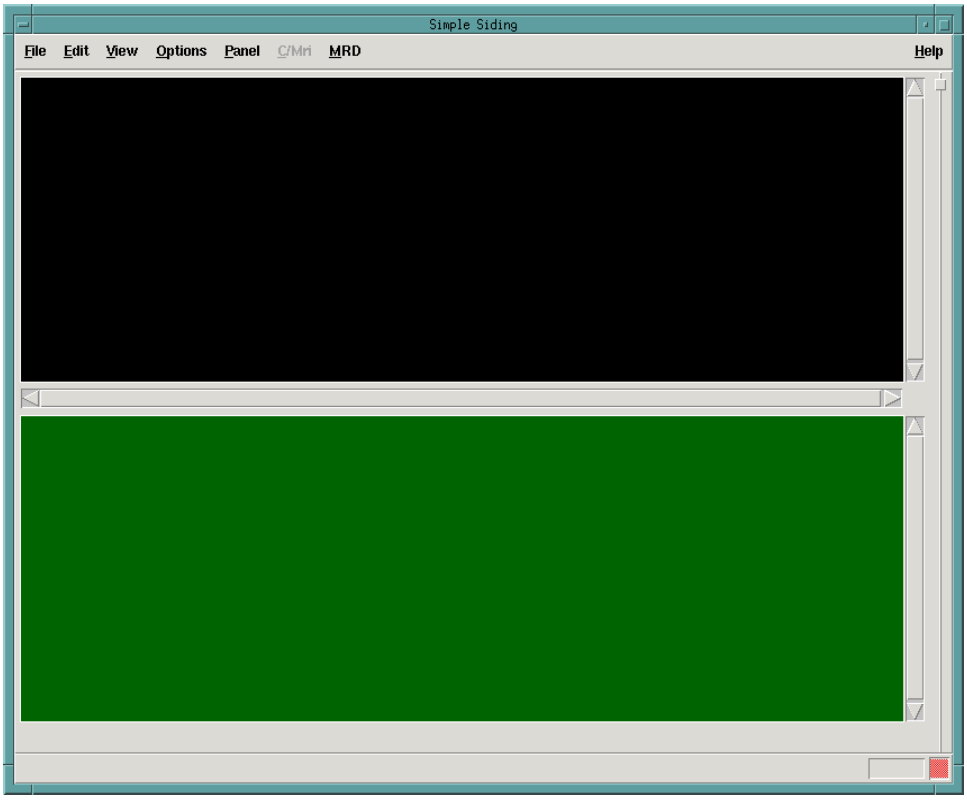


Figure 18.2 Initial blank panel

Now we can start adding track work and control elements. But first a brief discussion about how things are structured. First of all every object has a unique name and every object is in a named control point.
A "control point" is a collection of track work elements and control panel elements that relate to a single controlled feature, typically a turnout of some sort. The control point usually includes a code button, which is a button that initiates some change in the track work (turnouts, signals, etc.), based upon the settings of one or more control panel elements. In this tutorial we will be creating four control points, **CP1**, **CP2**, **Main**, and **Siding**. **CP1** is the turnout at the Western (left) end of the siding, **CP2** is the turnout at the Eastern (right) end of the siding, **Main** is the mainline trackage, and **Siding** is the siding track. The **Main** and **Siding** control points won't have any control panel objects and are only being used to contain the simple track elements. These are essentially "dummy" control points and are just being used as containers for track work that does not contain any centrally controllable track work.

First we will create turnout 1 (named **Switch1**) by selecting `Add Object` on the `Panel` menu, which gives us the `Add Panel Object to panel` dialog box, shown below. See Section Adding, Editing, and deleting elements to CTC Panel Windows for more information.

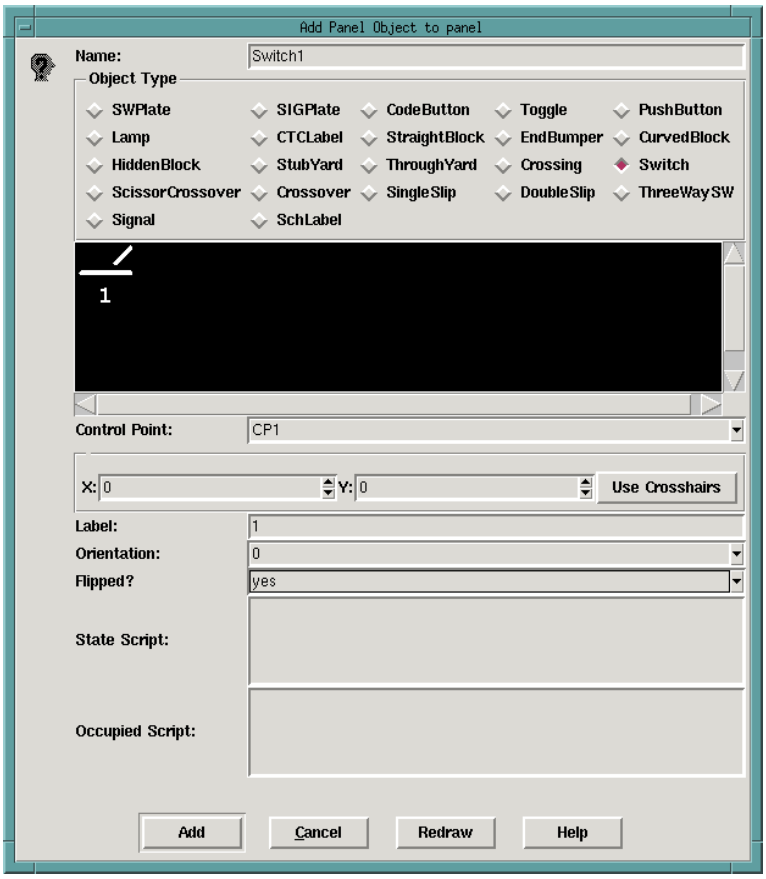


Figure 18.3 Creating Turnout 1

We will "flip" the turnout to give it the proper orientation. Turnouts can be flipped and can also be rotated to one of eight positions (45 degree increments). We will use the cross hairs to roughly position the turnout, as shown below.

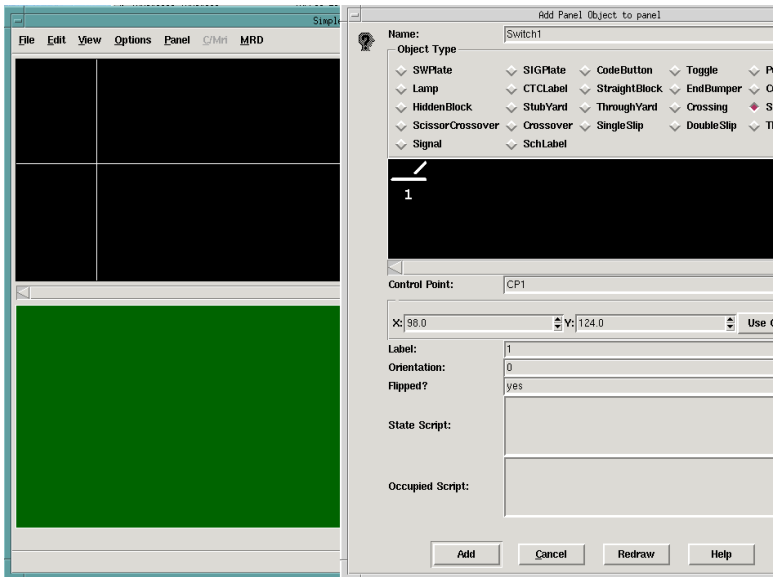


Figure 18.4 Positioning Turnout 1

Clicking the `Add` button places the turnout on the track work schematic, as shown below.

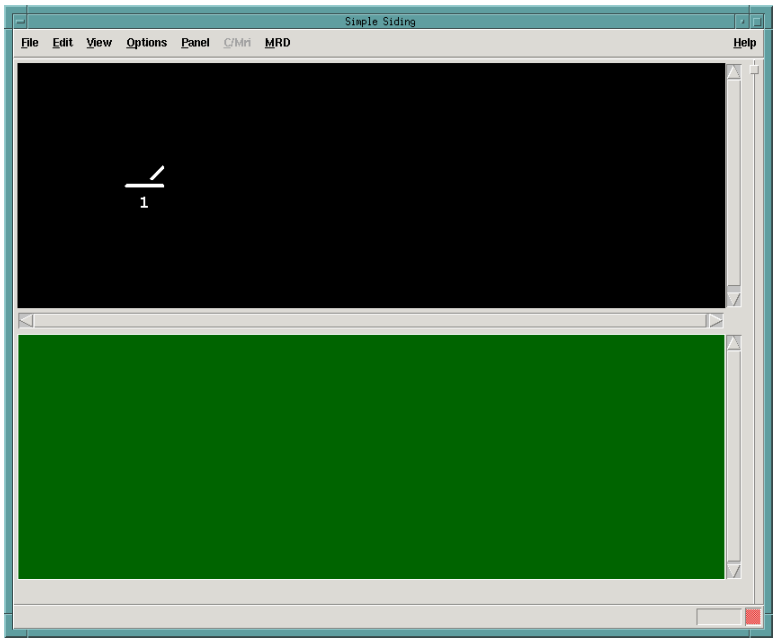


Figure 18.5 Turnout 1 placed on the panel

You can fine tune the location of the object by making small changes to the X and Y coordinates after you have roughly placed the object using the cross hairs. You can always go back and edit an object by using the `Edit` Object menu item on the `Panel` menu and then selecting the name of the object to edit.

Next, we will add a switch plate (named **SwitchPlate1**), again by selecting `Add Object` on the `Panel` menu, again using the @ Add `Panel Object to panel` dialog box, shown below.

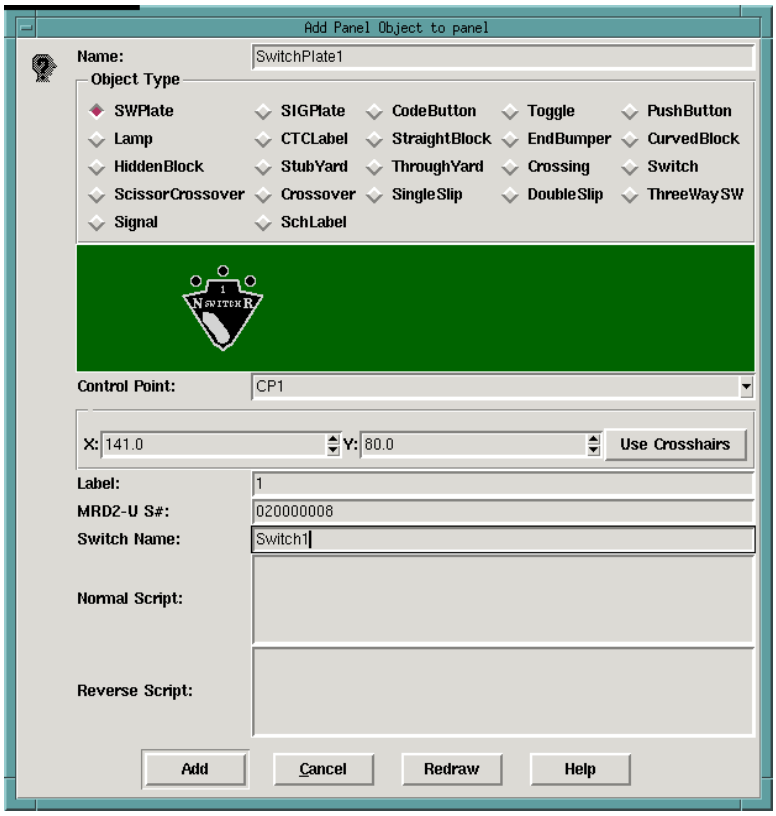


Figure 18.6 Adding a Switch Plate

We will enter the name of the turnout it controls (**Switch1**) and the serial number of the MRD2-U board that will be controlling the Switch-It board powering the switch motor. Again we will use the cross hairs to place the switch plate. The result is shown below.

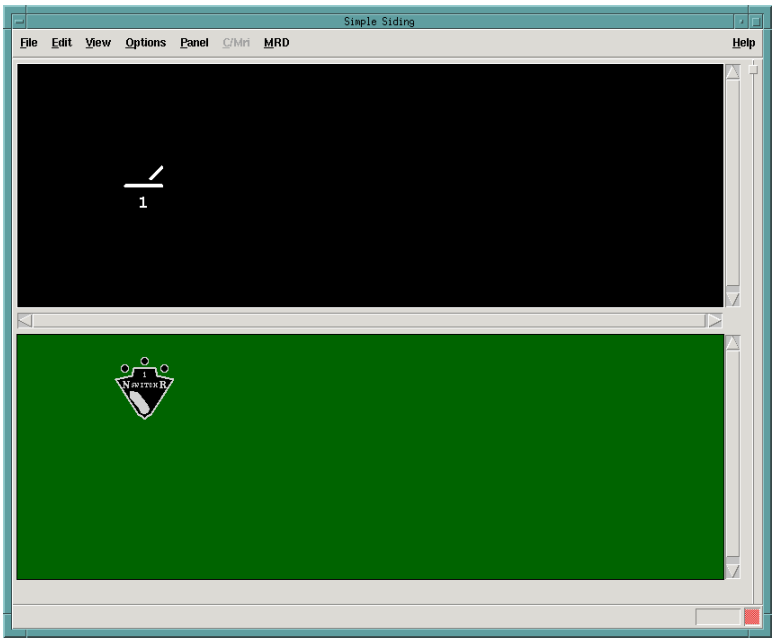


Figure 18.7 Switch Plate 1 placed on the panel

Finally, we will add a code button, as shown below.

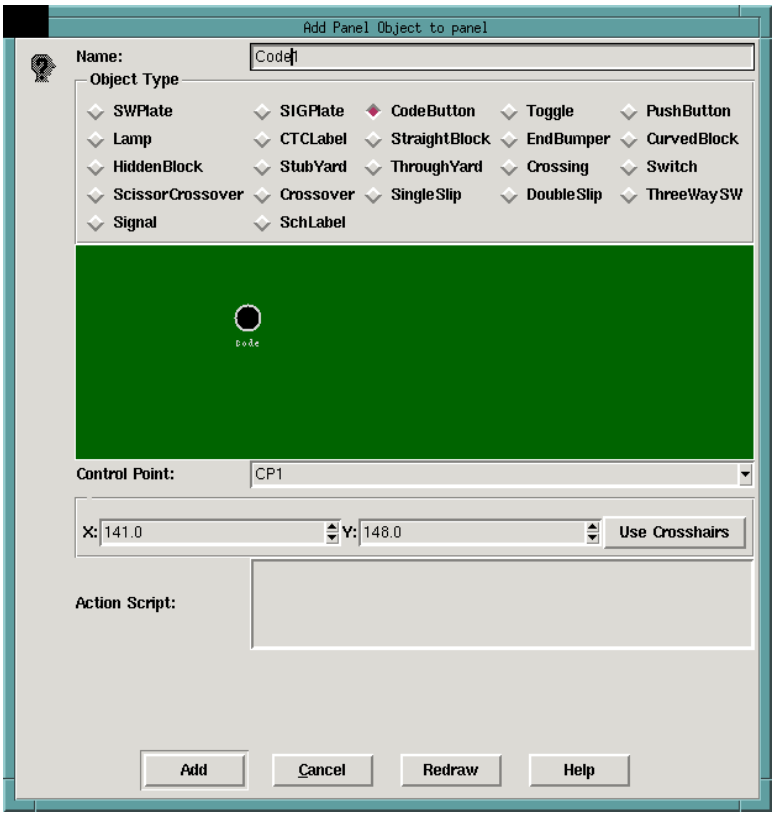


Figure 18.8 Adding a code button

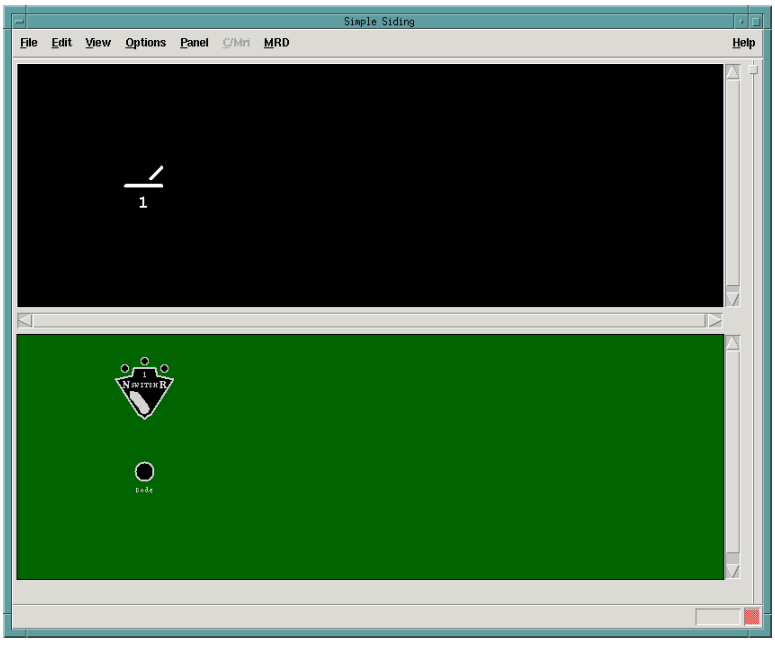


Figure 18.9 Code button 1 placed on the panel

We repeat this process to add the mainline, the siding, and the second turnout, with its switch plate and code button.

Place the second turnout next, then add the mainline and the siding tracks. Once the turnouts have been placed, the locations of the endpoints of the straight track sections are easy to select. Finally we get the panel shown below.

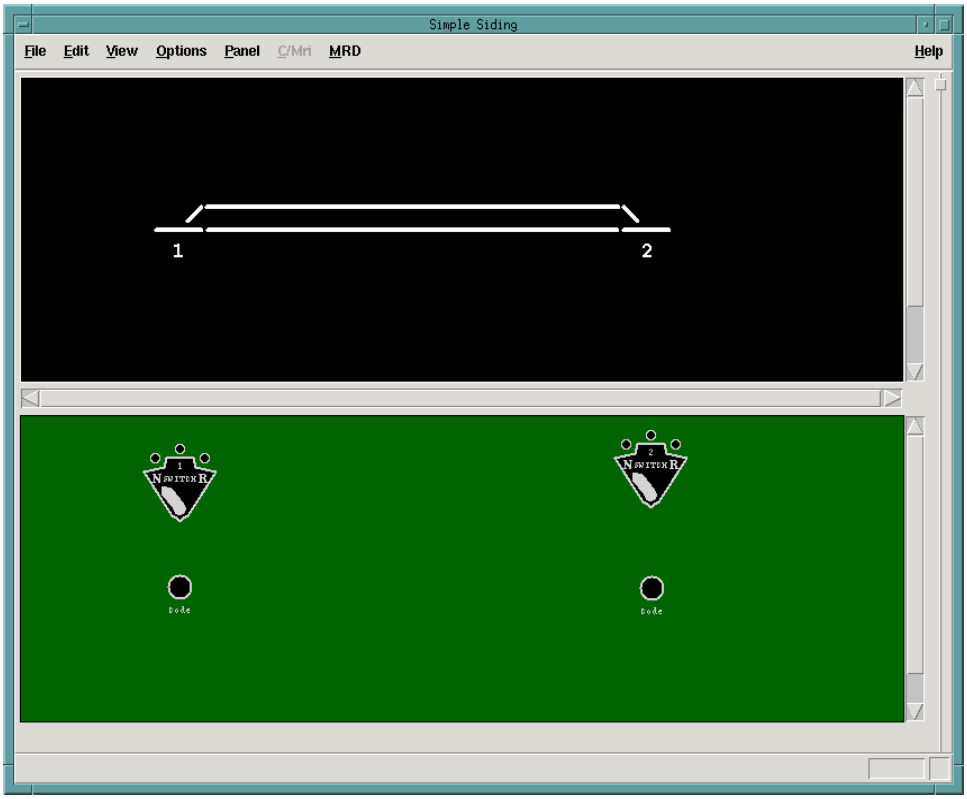


Figure 18.10 The completed panel

Once the panel has been completed, we can use the `Wrap As` menu item under the `File` menu to create a "wrapped" version of the generated program. This is a self-contained, stand-alone executable program that implements the CTC panel. See Section Wrapped CTC Panel Programs for more information.

18.2 A LCC Example

In this tutorial we will transfer a layout designed in XtrackCAD to a dispatcher CTC panel. XtrackCAD implements five type of "Layout Control" objects. These are objects that can be manipulated and stored with a layout file that hold layout control information (called scripts). These objects are:

- Blocks. Blocks are named groupings of track that have a script that determines block occupancy.
- Switch Motors. Switch Motors are named objects associated with a turnout and have three scripts: a normal action script, a reverse action script, and a point sense script. The normal action script activates a switch machine to align the points to the normal position (generally aligned to the main route) and the reverse action script activates a switch machine to align the points to the reverse position (generally aligned to the spur route). The point sense script returns a value indicating the current alignment of the points (eg normal or reverse).

- Signals. Signals are devices that display various aspects that indicate train movement permissions. Signals have a location, an orientation, a number of heads (one, two, or three), and one or more aspects, which have a name and a script.
- Sensors. Sensors are just single on or off state. A sensor could be a push button switch or some sort of device that senses something on the layout (generally not block occupancy, since that is already handled by the block object).
- Controls. Controls are just a single actuator or other on or off device (like a lamp). Could be some track side animation (like a log loader or a crossing gate).

XtrackCAD imposes no particular syntax for the scripts. For layouts using LCC and Dispatcher, the scripts are defined as one or two LCC event numbers. A LCC event number is a 64-bit number formatted as eight pairs of hexadecimal digits separated by periods. For actions, it is a single event id that is consumed by the actuator (eg switch machine or signal). For sensors (eg block detection sensors, point sensors), it is a pair of event numbers separated by a colon. These event numbers are produced by the sensor for each of two states. For blocks, the first event number is produced when the block becomes occupied and the second event number is produced when the block becomes unoccupied. For point sense scripts the first event is produced when the points become normal aligned and the second event is produced when the points become reverse aligned.

The layout we will add is a simple passing siding on a main line as shown below.

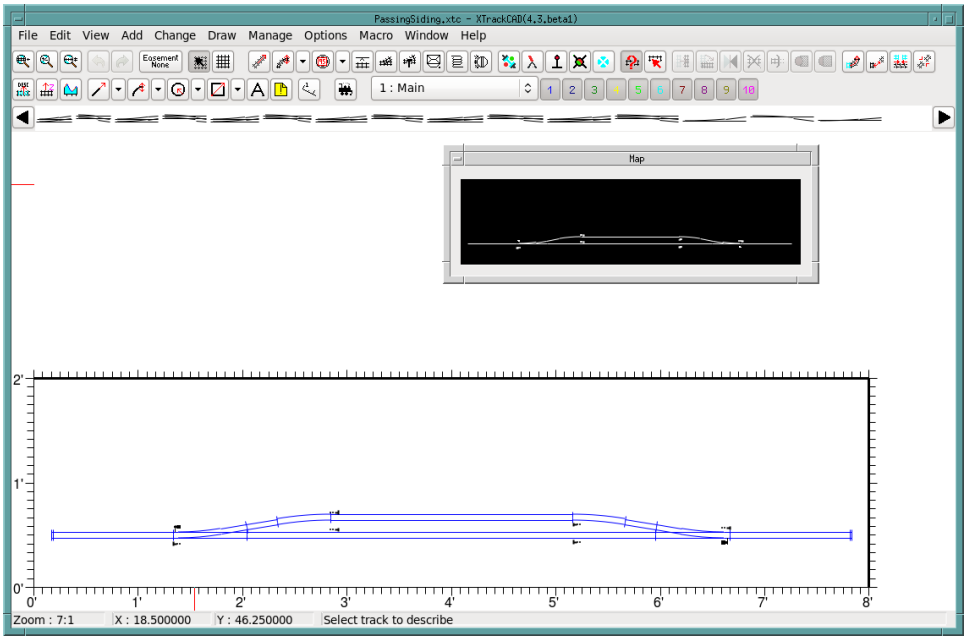


Figure 18.11 Simple passing siding in XtrackCAD

We have already filled in the control elements for this layout in the XtrackCAD layout file.

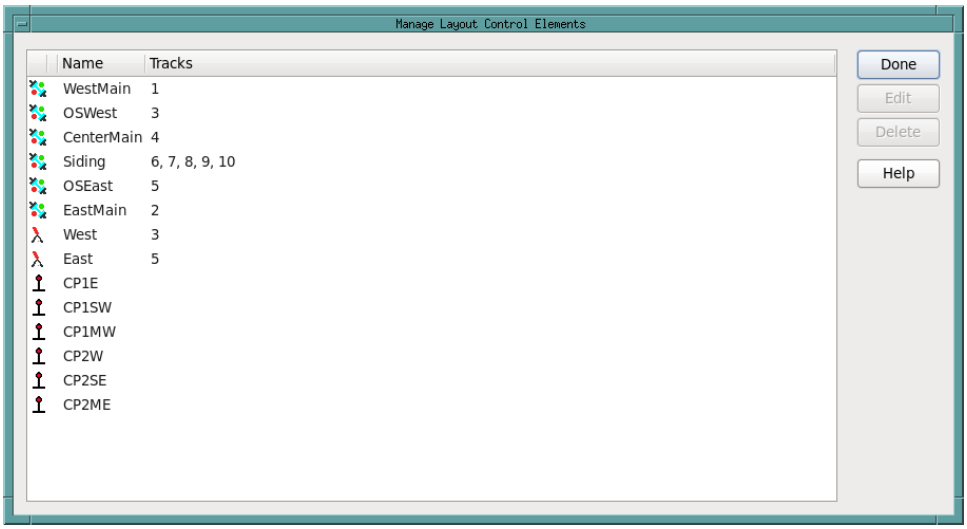


Figure 18.12 Passing Siding Layout control elements

After loading the XtrackCAD file into the Dispatcher, the dispatcher shows a graph of the layout, detailing the various track and control elements and their connectedness.

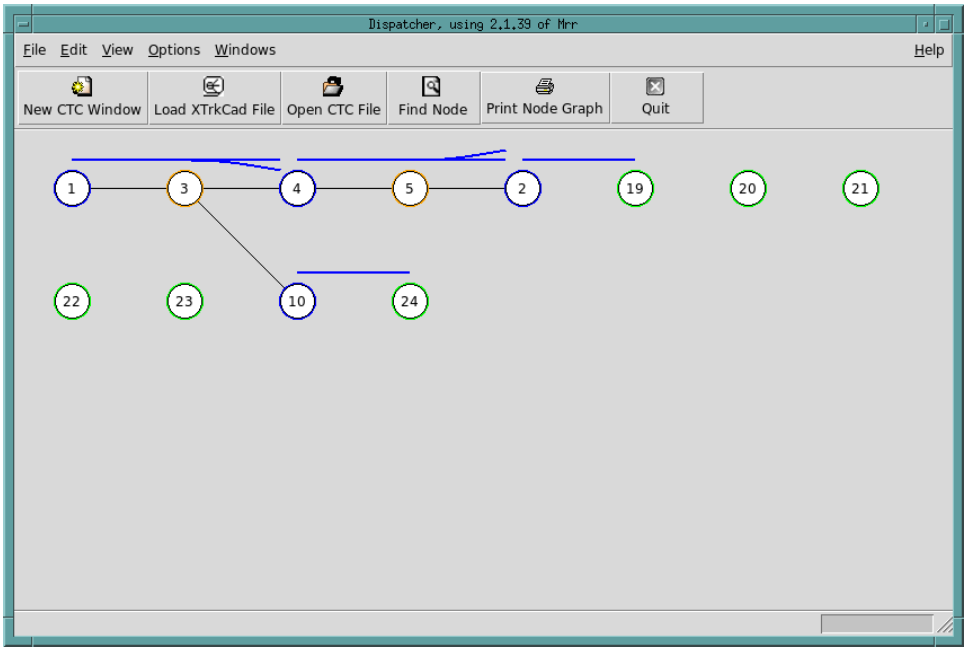


Figure 18.13 Passing Siding Layout as a graph in Dispatcher

Next we will create a fresh CTC Panel by clicking on the "New CTC Window" button. This gives us a "New CTCPanel" dialog box.

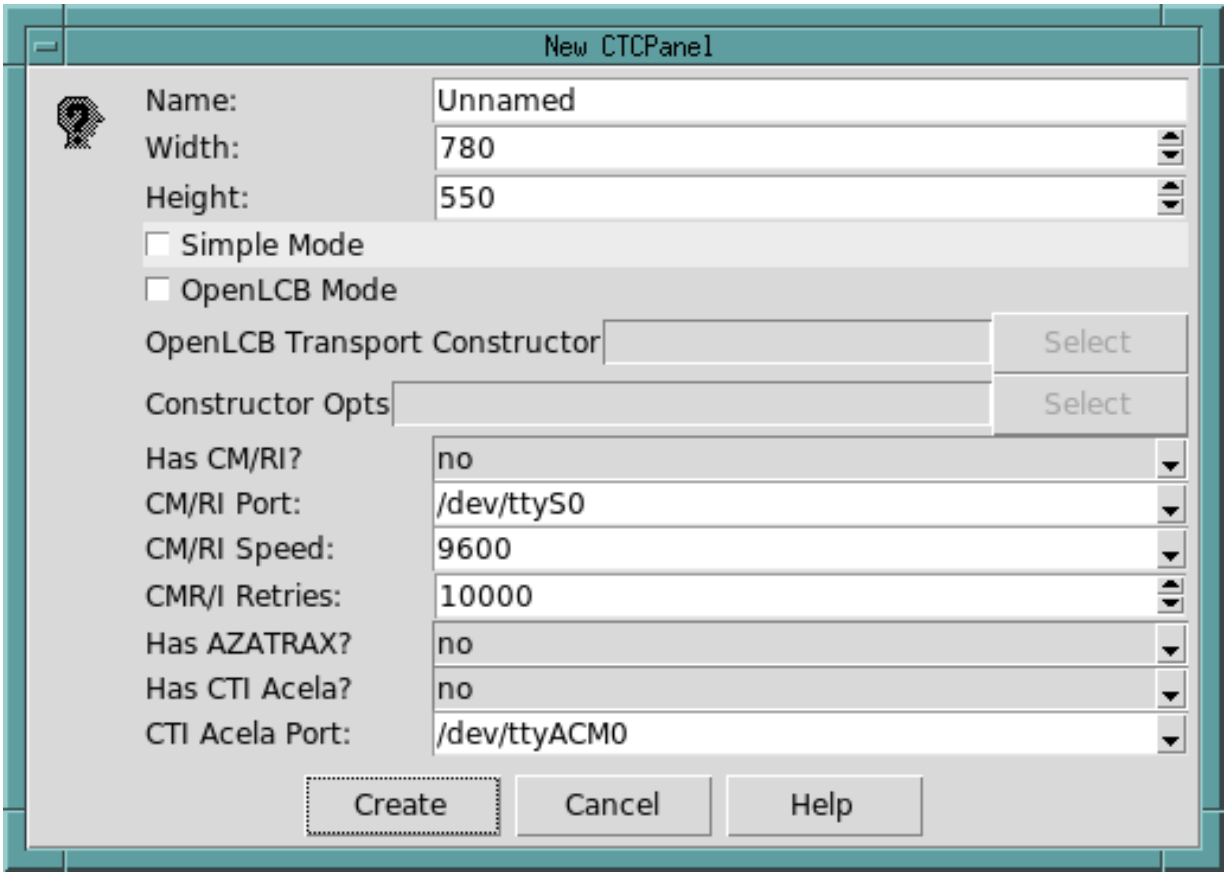


Figure 18.14 New CTCPanel dialog

We will give the CTC Panel a name (Passing Siding), select the OpenLCB Mode checkbox, and fill in the OpenLCB transport constructor and its opts.

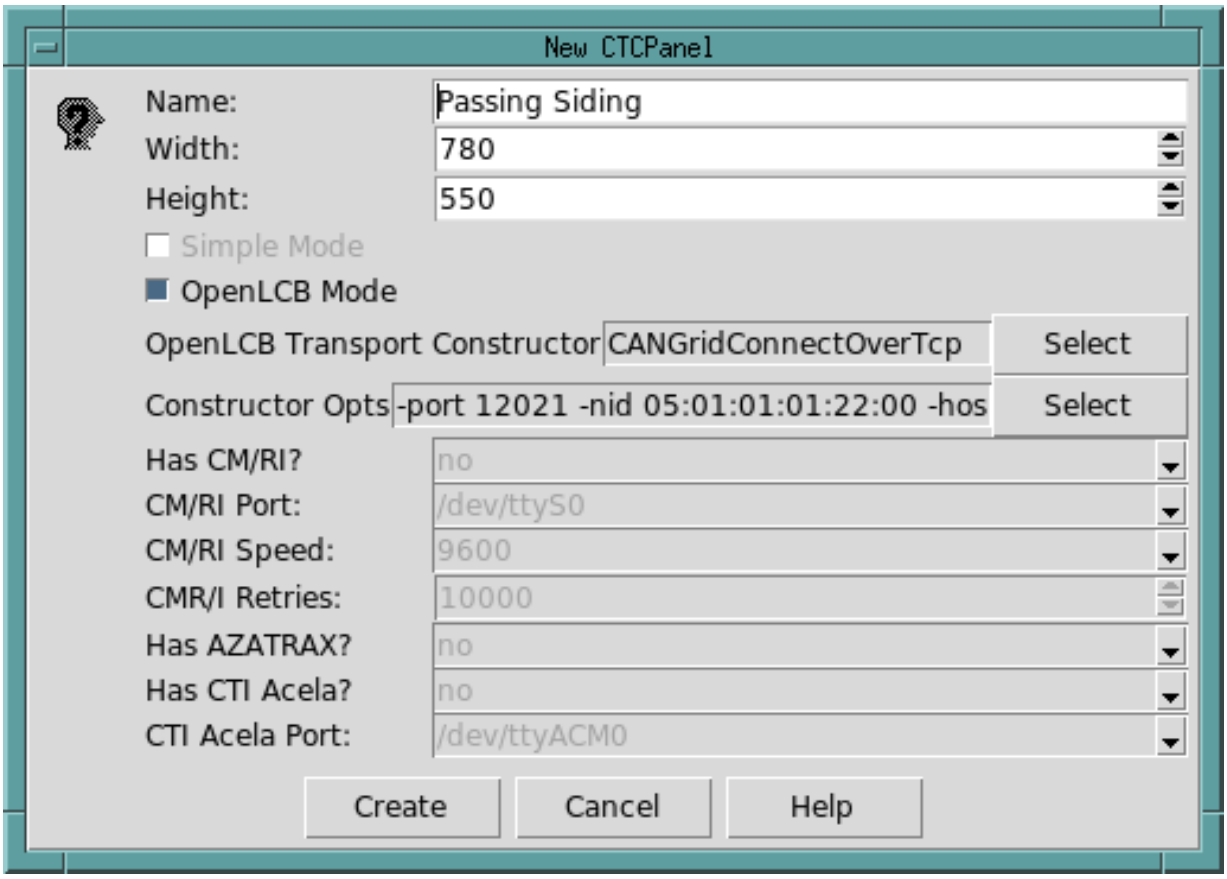


Figure 18.15 New CTCPanel dialog, updated

After clicking the Create button we have a new fresh CTC Panel window.

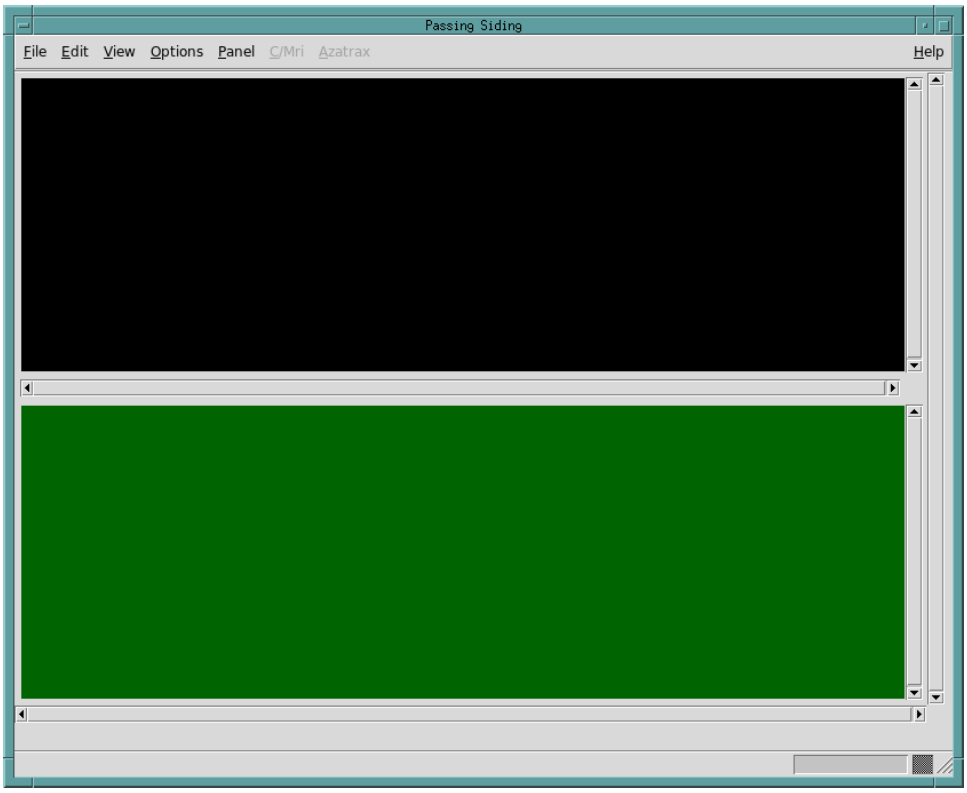


Figure 18.16 Empty Passing Siding CTC Panel

First we will transfer track segment 1, the western most segment of the main line just before the siding. We will use the right button to get the context menu and then select "Add To Panel" on this menu.

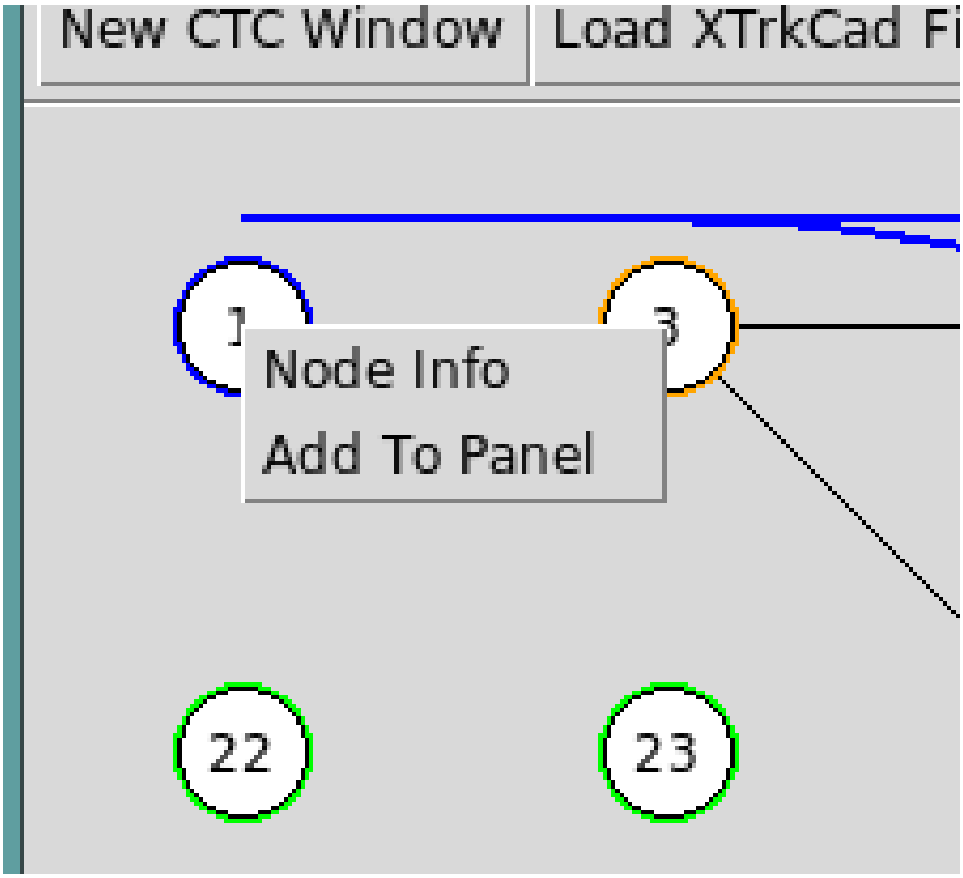


Figure 18.17 Context menu for track element 1

This gives us an "Add Panel Object to panel" dialog box. Notice how various fields are preloaded. We will shortly change a few things: remove the unneeded label, add a control point, and position the track on the schematic using the cross hairs.

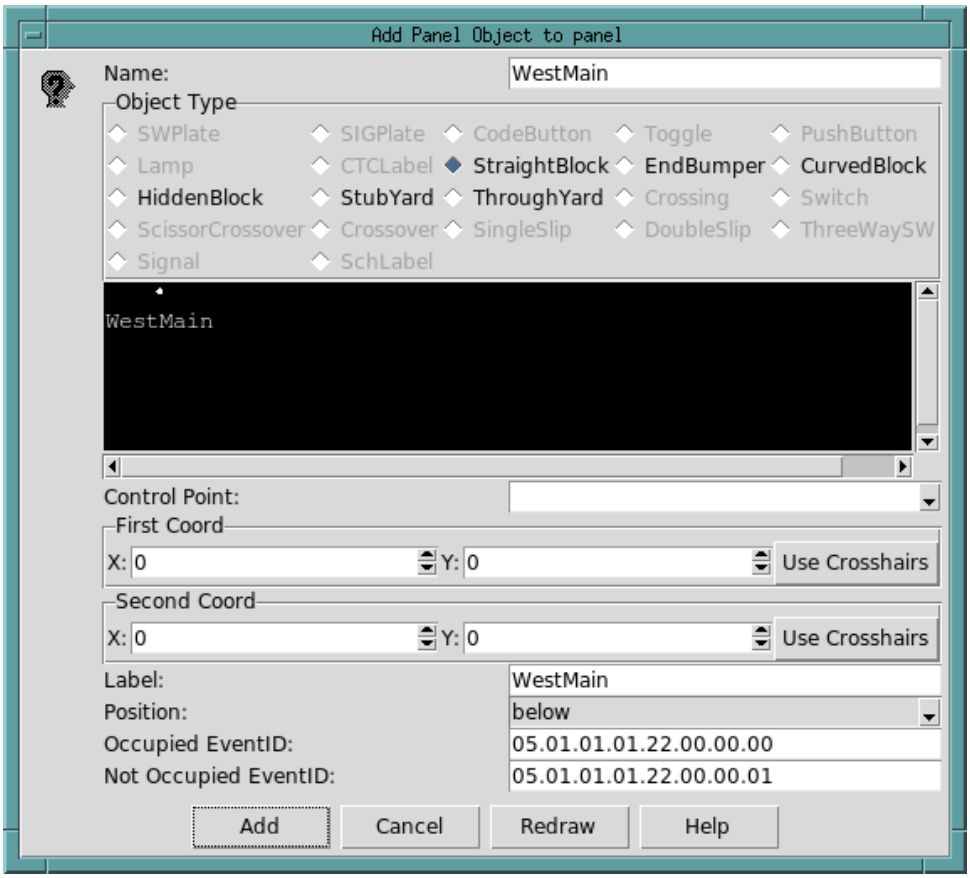


Figure 18.18 Initial Add Panel Object to panel dialog box for segment 1

And after some minor updates, we have this:

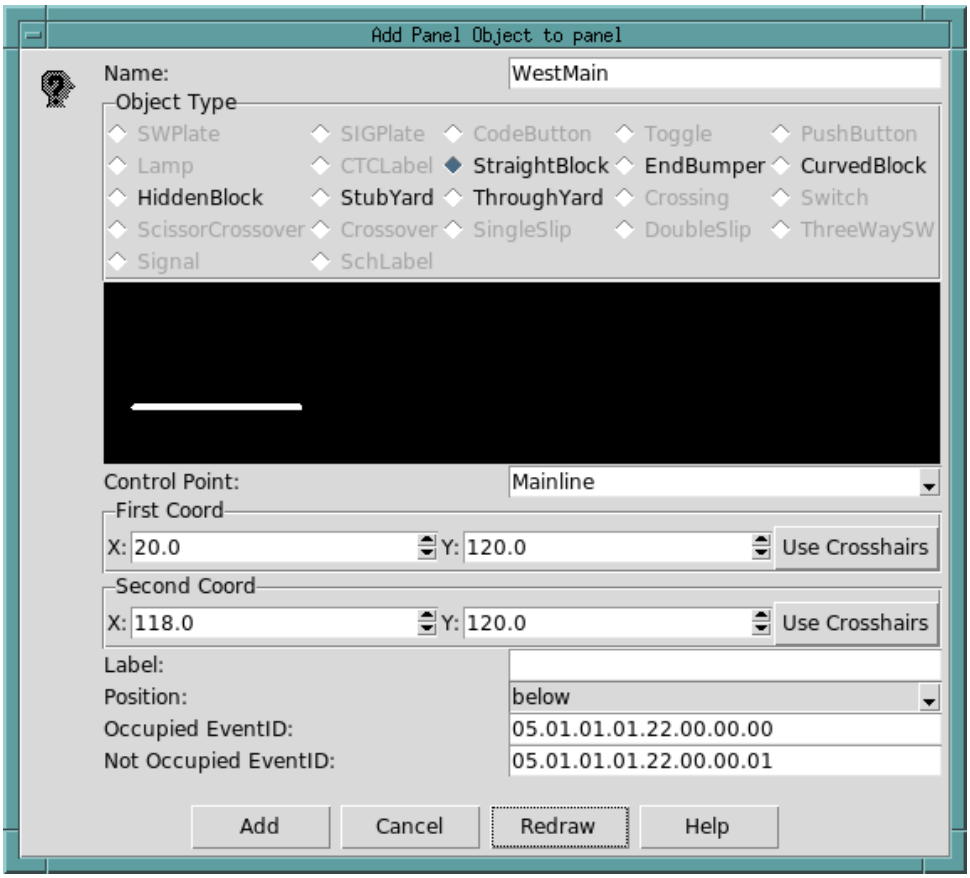


Figure 18.19 Updated Add Panel Object to panel dialog box for segment 1

Next we will add the western turnout. This is track segment 3. Again we right click on the segment node on the Dispatcher main window and select "Add to Panel". Note how most of the fields are filled in from the layout file.

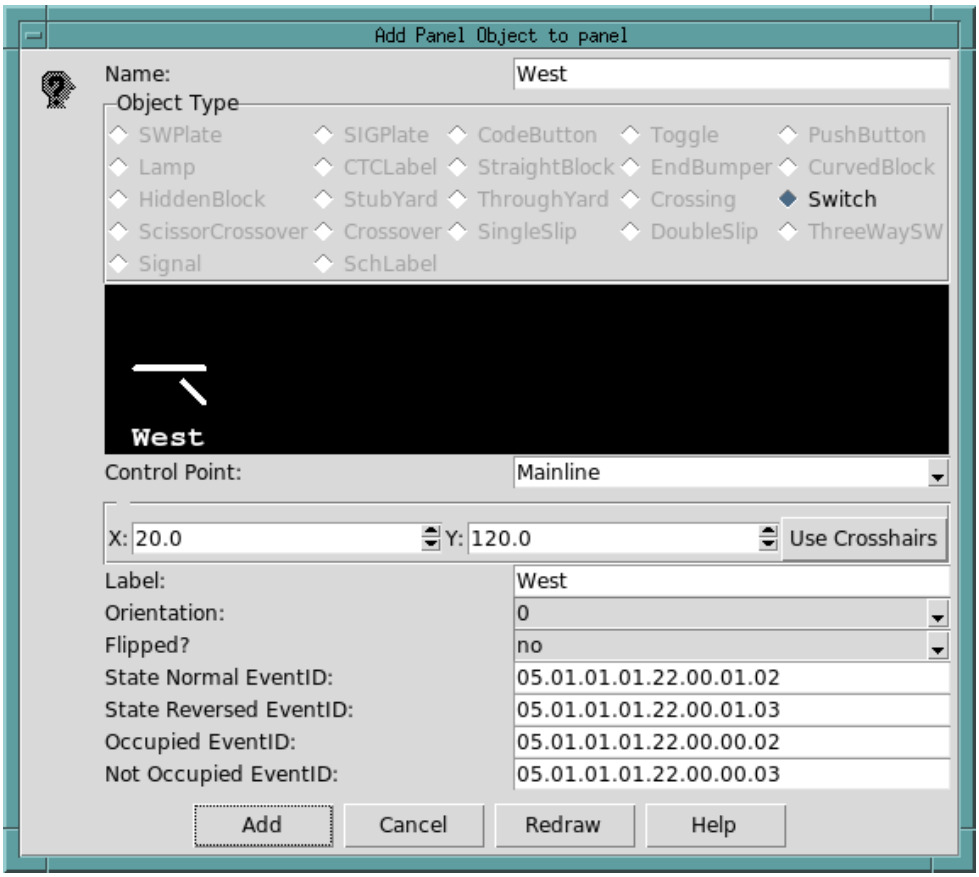


Figure 18.20 Initial Add Panel Object to panel dialog box for segment 3

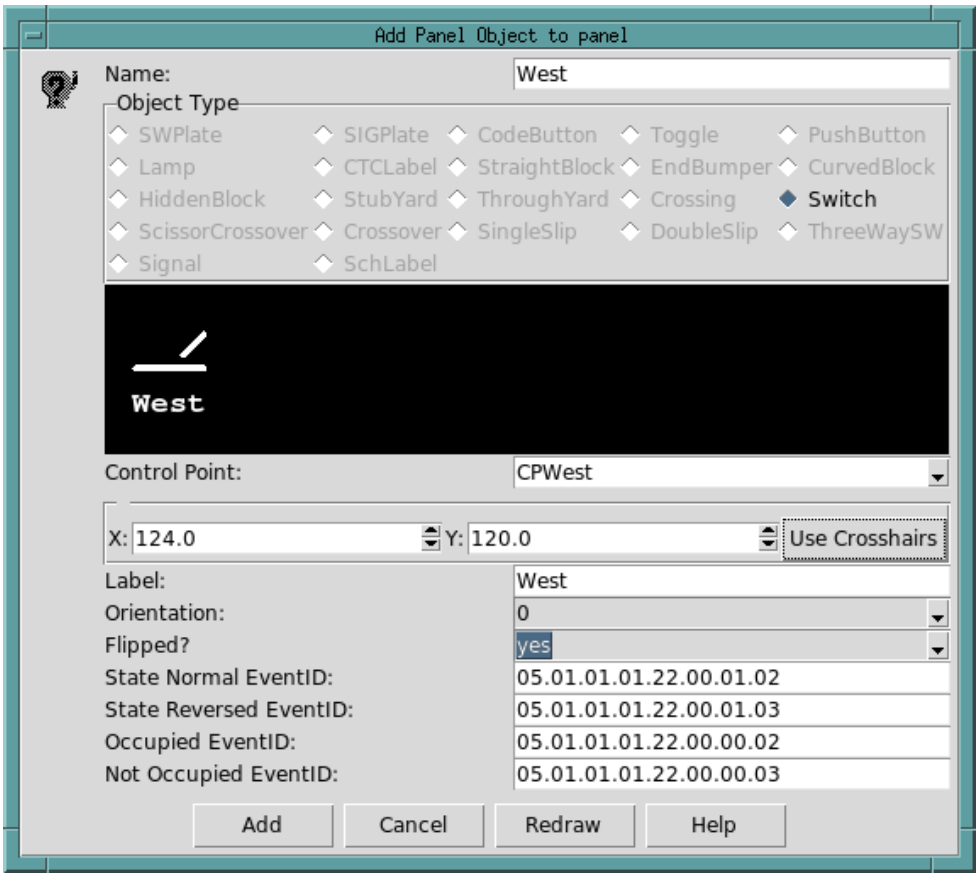


Figure 18.21 Updated Add Panel Object to panel dialog box for segment 3

Because this is a turnout with a switch motor, we will also add a switch plate. The only additional work needed is positioning it.

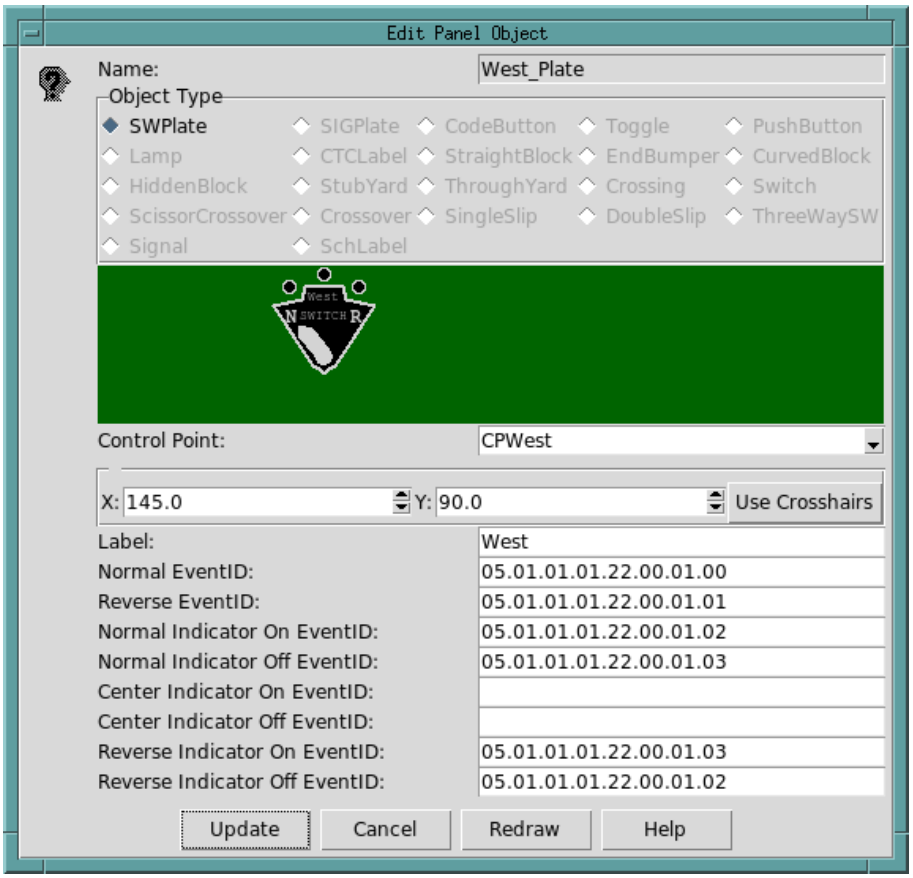


Figure 18.22 Updated Add Panel Object to panel dialog box for turnout switch plate

Our panel is now like this:

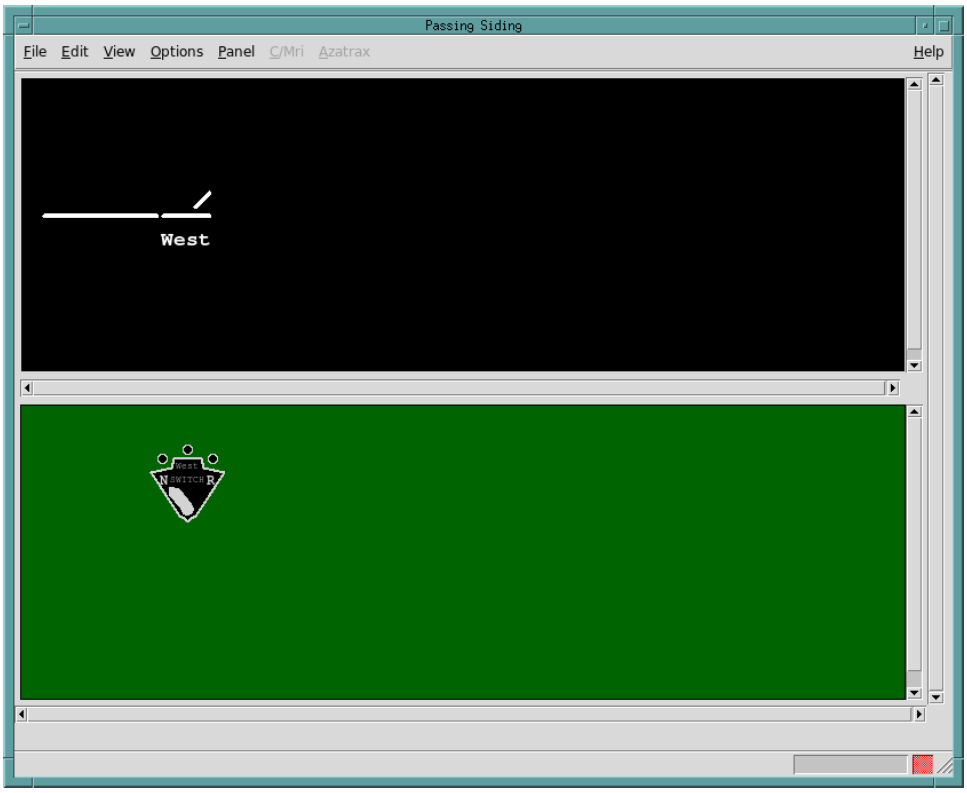


Figure 18.23 Updated CTC Panel

Finally we will add one of the signals, CP1E, which is node number 19.

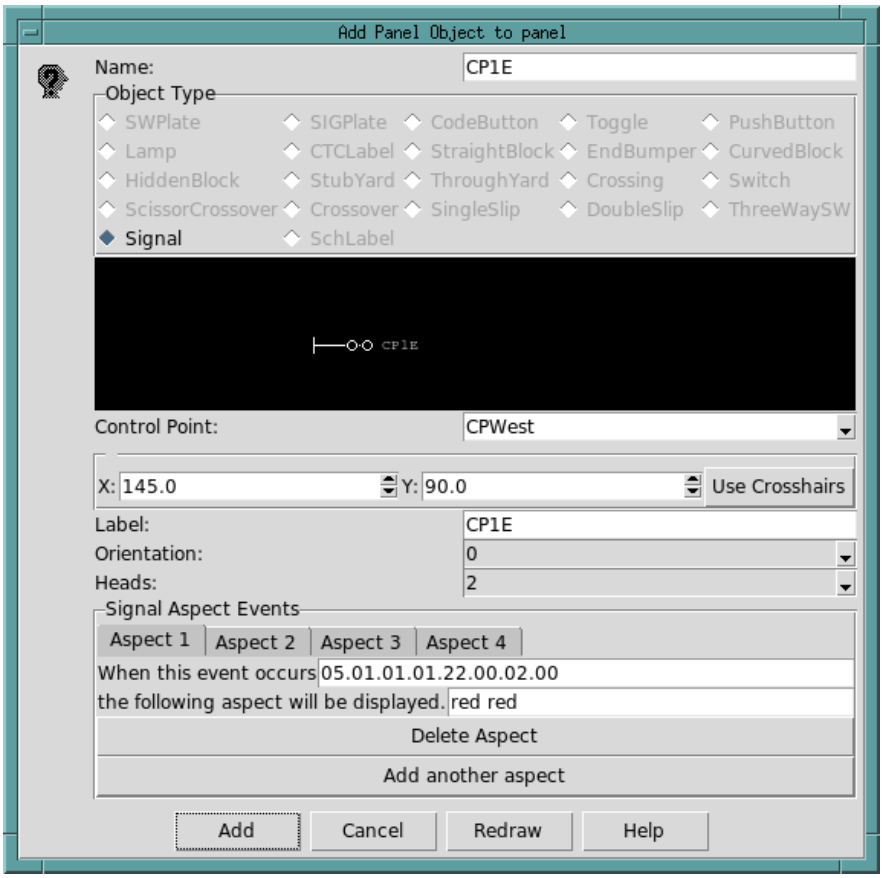


Figure 18.24 Initial Add Panel Object dialog box for CP1E

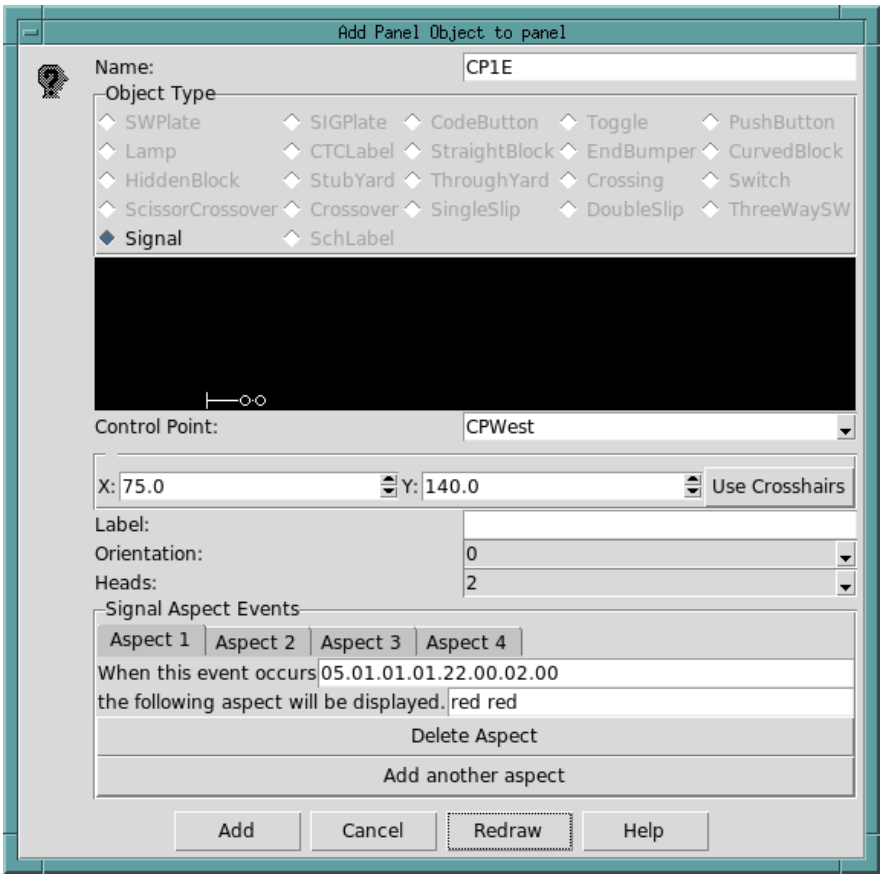


Figure 18.25 Updated Add Panel Object dialog box for CP1E

This gives us this panel:

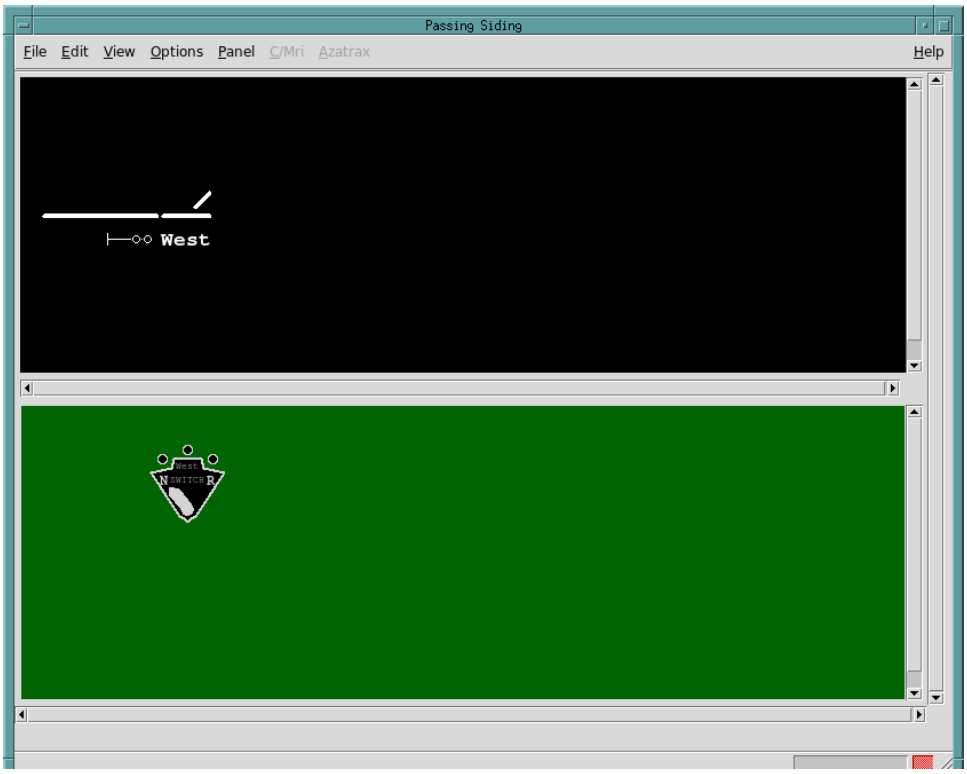


Figure 18.26 Panel with signal CP1E added.

The remaining track elements and switch plates can be added by repeating these steps.

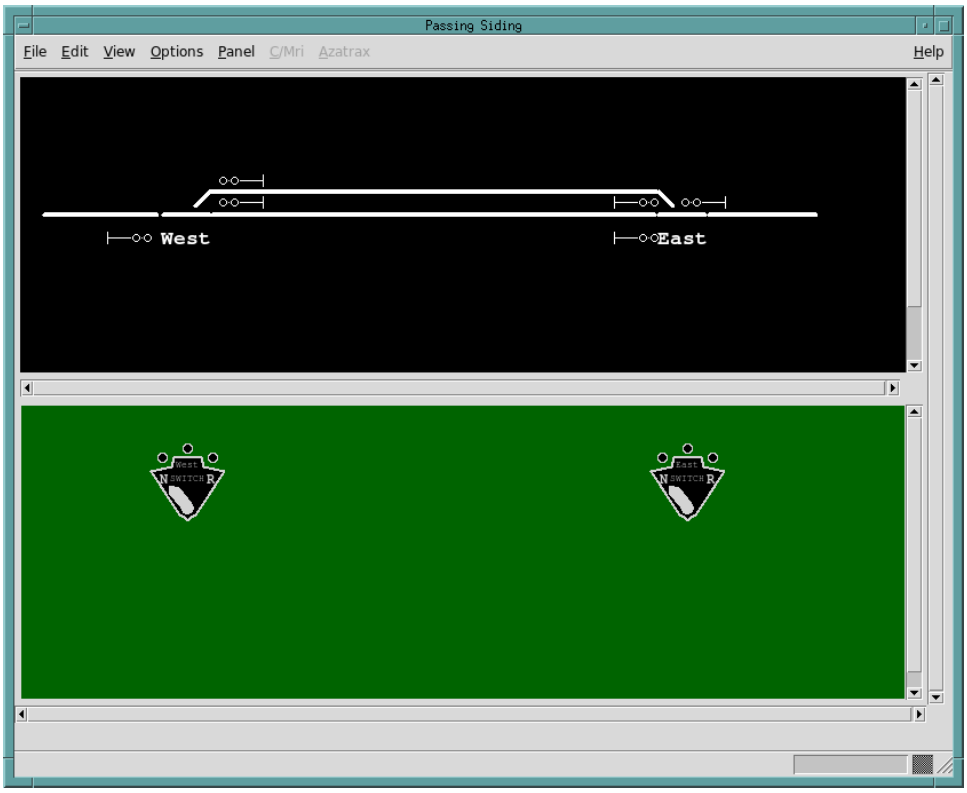


Figure 18.27 Panel with all elements from the layout file added.

Finally, the signal plates and code buttons can be added using the Panel menu on the CTC Panel.

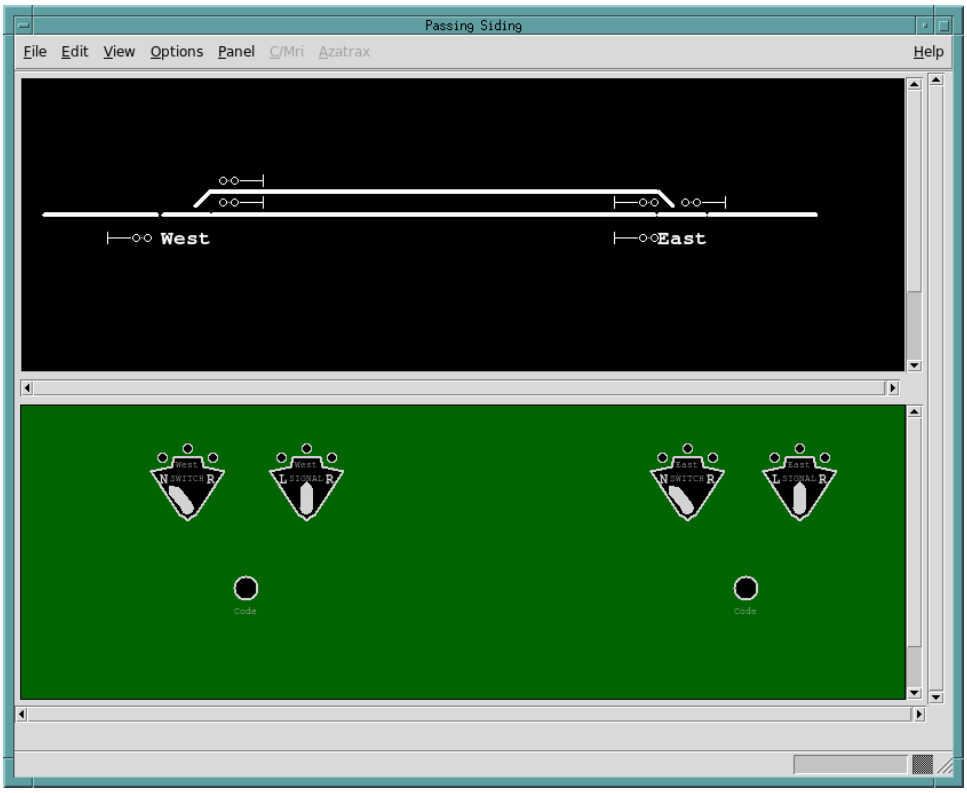


Figure 18.28 Completed panel.

Now we can save the file and wrap it into a ready to run executable.

Chapter 19

Dispatcher Reference

The Dispatcher program is used to create computerized CTC (Centralized Traffic Control) panels, to be used by dispatchers as part of a CATC (Computer Assisted Traffic Control) system to manage traffic flow for a model railroad. [1] A computerized CTC panel typically contains a track work schematic and a collection of control elements (such as switch plates, signal plates, toggle switches, push buttons, etc.) that control the track work and track side signals. In addition to creating and editing CTC panels, the Dispatcher program can read in an XTrackCAD layout file and create a compressed graph of the track work and this graph can be used as a guide while creating CTC panels.

The Dispatcher program can use a Layout Control Database to manage the various layout control elements. This database is also used by the OpenLCB (see OpenLCB Program Reference) and the Offline LCC Node Editor (see Offline LCC Node Editor Reference) programs.

19.1 Main GUI Screen

The main GUI window of the Dispatcher program is shown below. It consists of a standard menu bar, a tool bar, and an area for track work graph display. A track work graph is something computer scientists call a "directed graph". A directed graph is a data structure consisting of nodes, with links to other nodes. In this case each node is a piece of track work and the links are the connections between pieces of track work and thus indicate the adjacency of track work nodes (and how the pieces of track work interconnect with each other). In the case of simple track work (such as straight sections or curved sections), there are one (if the section "dead ends") or two links and in the case of complex track work (such as turnouts and crossings), there are three or more links. The graph is "compressed", where adjacent pieces of simple track work are consolidated into single nodes.

[1] It is also possible to use the Dispatcher program to create the artwork for a “manual” CTC panel using mechanical switches mounted on a panel.

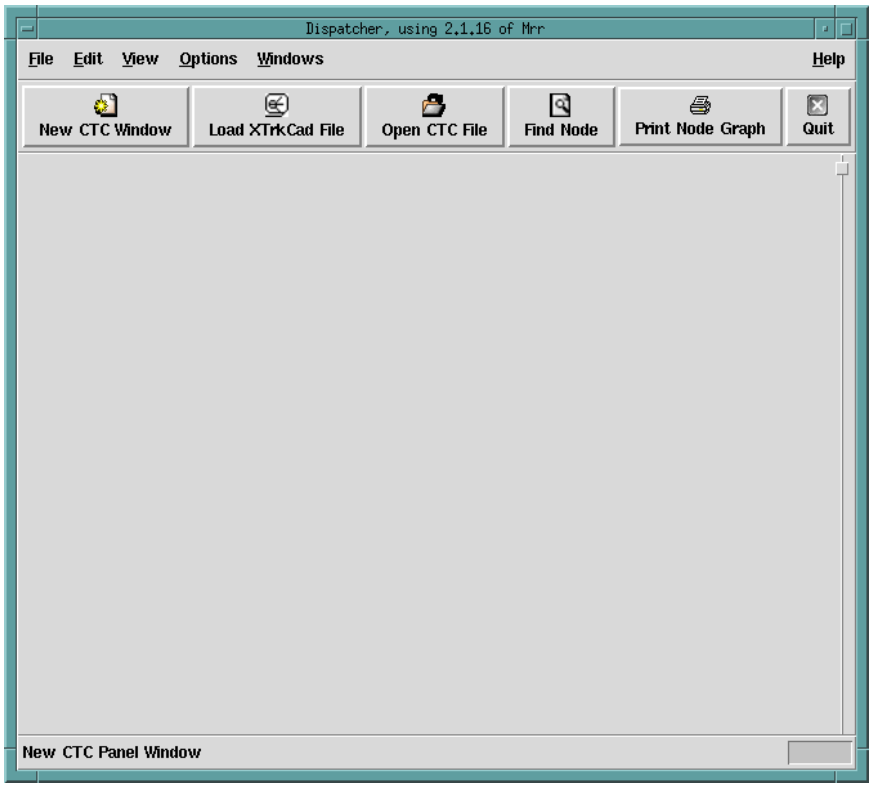


Figure 19.1 Main Dispatcher Window

19.1.1 Track work Node Graphs

Once the compressed graph is built (upon loading a layout file), it is displayed on the main GUI screen as numbered circles (nodes) connected by lines (links). A graphic of the piece of track work is also drawn. Left clicking on a node displays a pop-up containing information about the track work node. Right clicking on a node pops up a menu of actions involving the node. [2] Left clicking on a link displays a pop-up containing information about the link. The node numbers are the track element numbers assigned by XTrackCad.

The outlines of the nodes are color coded:

- Blue for block nodes.
- Orange for turnouts with switch motors.
- Green for signals.
- Light Green for sensors.
- Light Blue for controls.
- and Black for everything else.

[2] Currently two menu items are defined, one to display the node info and the other to add it to a panel.

19.1.1.1 Loading a Layout

A XTrackCAD layout file is loaded either with the `Load XTrkCad File` tool bar button or with the `Load...` item on the `File` menu. It is also possible to load a XTrackCAD layout file during program start up using the `-xtrkcad` command line option.

In all cases, the layout's track work is loaded as a track work graph and displayed in the track work graph display area. Dispatcher then asks if XTrkCad itself should be started. This might be useful as it gives a display of the actual layout on your screen, which can be referred to when creating CTC panels.

19.1.1.2 Finding Nodes

A node can be "found" by its number. Finding a node scrolls the node graph to make the node visible and the node is highlighted.

19.1.1.3 Printing Node Graphs

A node graph can be printed to provide a hard-copy reference of the node graph.

19.1.2 Creating a new CTC Panel

A new CTC Panel is created with the `New CTC Window` toolbar button or the `New CTC Panel Window` entry under the `File` menu. A New CTC Panel Dialog, as shown below, is displayed. This dialog box asks the user for some basic information about the new panel. This includes the name (title) of the panel, its initial width and height, whether it connects to a C/MRI bus and information about that bus, as well as whether it uses Azatrax devices or if it using LCC (OpenLCB mode). [3]

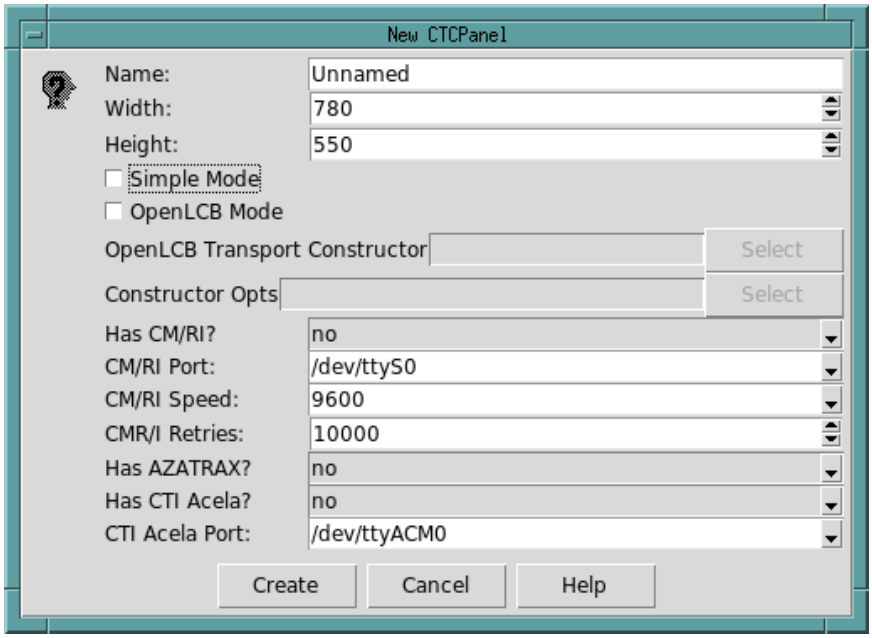


Figure 19.2 New CTC Panel Dialog

[3] MRD2-U, MRD2-S, SR4 or other devices made by Azatrax.

There is also a check button to select "Simple Mode". "Simple Mode" simply means that the panel will be a simple one with canned code to use Azatrax USB devices to actuate switch machines, either directly or via NCE's Switch-It (or similar) units. The canned and auto-generated code associates a Switch Plate with a switch and associates a Code Button with 1 or more Switch Plates and Signal Plates. [4] In "Simple Mode", all of the UI elements relating to writing Tcl code are disabled and all of the Tcl code is completely generated by the Dispatcher program. All the user does is place the track work elements on the schematic and the control elements on the control panel and provide the serial number of the MRD2-U or other Azatrax devices that are being used to control turnouts and the names of the turnouts being controlled.

It is also possible now to build a CTC panel that connects to an OpenLCB network (either on a CAN bus or a Tcp/Ip network) and use the Event Exchange protocol to connect the schematic track work and control elements on the CTC panel with physical OpenLCB nodes with I/O pins connected to sensors, actuators, and signals on your layout. Selecting "OpenLCB mode" and supplying the transport information allows for this. All of the action code for the panel objects are replaced with OpenLCB events and there is no "Main Loop" (as discussed below).

What is actually created is a Tcl/Tk program that uses parts of the library of Tcl/Tk and C++ code included with the Model Railroad System that will implement a CTC Panel, which contains two display sections: a track work schematic and a control panel. The track work schematic is in the upper half of the panel and has a black background with white (red when occupied) track work. Signals with one, two, or three heads can be added to the track work schematic. The control panel is in the lower half and has a dark green background. The control panel can have switch plates, signal plates, toggle switches, push buttons, code buttons, and indicator lamps.

The CTC Panel can be used by a dispatcher using a computer screen and pointing device (such as a mouse or touch pad) to select and manipulate control elements. The track work on a CTC Panel will reflect the actual track conditions (occupied or not, signal aspects, and turnout states).

19.1.3 Opening an existing CTC Panel

Existing CTC Panel programs are the specifically formatted Tcl/Tk programs created by the Dispatcher program. They can be opened and edited using the `Open CTC File` toolbar button or the `Open...` entry under the `File` menu or specified on the Dispatcher's command line. There are three sections of the code that are "loaded" into the Dispatcher program: the collection of CTC Panel elements, the information about the (optional) C/MRI network or Azatrax USB devices, and the user code associated with the panel.

19.2 Configurable Options

The configurable options can be set or changed with the `Edit Configuration` menu entry under the `Options` menu. These configurable options can be saved with the `Save Configuration` menu entry under the `Options` menu and can be loaded with the `Load Configuration` menu entry. At present there are three configuration options: `Use External Editor`, which has a boolean value (true or false), `External Editor`, which is a command line that starts an external editor (the name of the file to edit is appended to this command line), and `Tcl Kit`, which is the name of the Tclkit file to use for the run time when wrapping panel programs (see Section Wrapped CTC Panel Programs).

[4] The Signal Plates don't do anything but change their panel lights.

19.3 CTC Panel Windows

A freshly created CTC Panel window is shown below. [5]

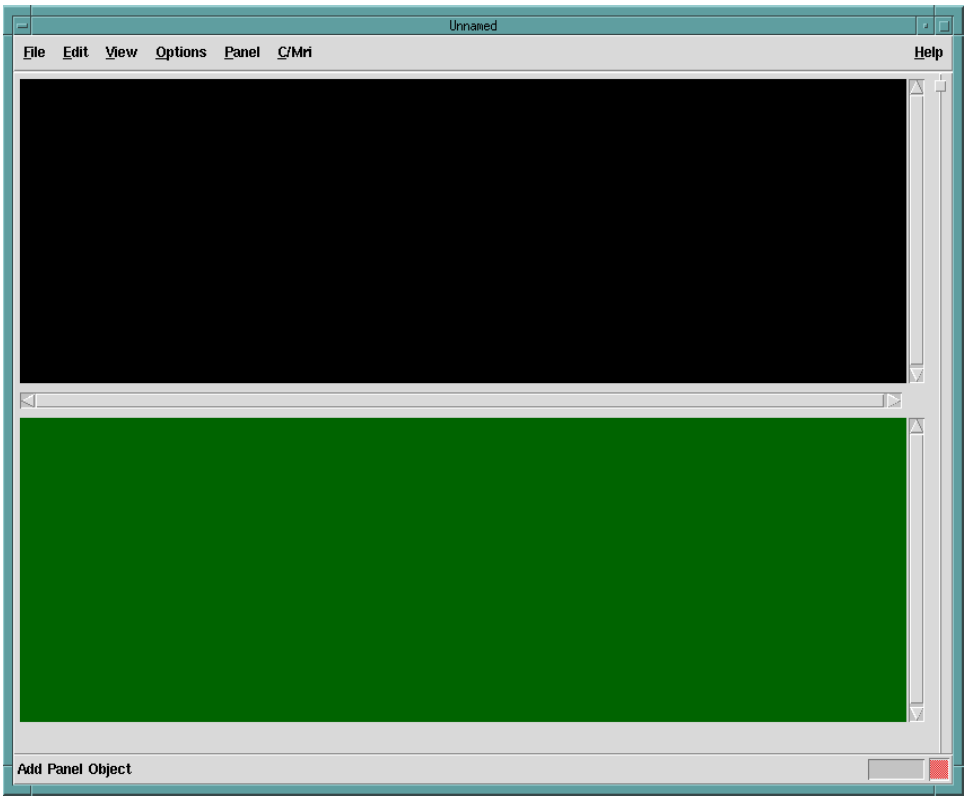


Figure 19.3 Empty CTC Panel Window

The pink square at the lower left indicates that the file is in a modified state and has not been saved to disk. Saving the file is done with the `Save` and `Save As...` menu items under the `File` menu. It is also possible to create a standalone executable program file using the `Wrap As...` menu item under the `File` menu (see Section Wrapped CTC Panel Programs for more information).

19.3.1 Menu items available when editing a CTC Panel Window

19.3.1.1 File menu

The `File` contains entries to create a new CTC Panel Window, load an XTrkCad file, open a CTC Panel Window, save the current CTC Panel Window, wrap the current CTC Panel Window, close the current CTC Panel Window, and exit. Attempting to close a modified CTC Panel Window will cause a save confirmation window, allowing you to save your work. There are also menu items to print the panel graphics as a PDF page or to export either the schematic or control panel as a bitmap image. These files can be printed and used as the artwork for a manual CTC panel using mechanical switches.

[5] When the program is run on its own, the Panel and C/Mri menus will be absent.

19.3.1.2 Edit menu

In addition to the standard edit menu entries, there are four extra entries: [6]

(Re-)Generate Main Loop This entry generates (or regenerates) the main loop. The basic loop read all of the input ports of all of the C/MRI nodes, invokes all of the track work elements, and then writes all of the output ports of all of the C/MRI nodes. The loop is an endless real time loop. It is necessary to "fill in" the logic of the CTC Panel.

User Code This entry opens an editor to edit the user code section of the CTC Panel program.

Modules This entry inserts selected helper modules into the user code section of the CTC Panel program. These are all in name spaces and are SNIT types:

- **Track Work types** This inserts two SNIT types, one for blocks (usable for simple track work) and one for switches (turnouts).
- **Switch Plate type** This inserts a SNIT type to handle switch plates.
- **Signals** This inserts SNIT types to help with signals:
  - **Two Aspect Color Light** Use this module if you are using two lamp or LED (red and green) signals. One, two, and three head signals are supported.
  - **Three Aspect Color Light** Use this module if you are using three lamp or LED (red, yellow, and green) signals. One, two, and three head signals are supported.
  - **Three Aspect Search Light** Use this module if you are using bi-color LEDs (red/green – either three lead or two lead) signals. One, two, and three head signals are supported.
- **Signal Plate type** This type handles Signal Plates.
- **Control Point type** Use this type for Code Button action code.
- **Radio Group Type** Use this type to collect a group of push buttons into an exclusive group where only one button is "on" at a time. Used to implement a software track selection matrix for a yard or terminal.

Additional Packages This entry inserts selected additional packages into the CTC Panel program. The available packages are:

- **XPressNet** This loads the XPressNet DCC package.
- **NCE** This loads the NCE DCC package.
- **Raildriver Client** This loads the Raildriver Client package.

19.3.1.3 View menu

The `View` menu contains entries to zoom in, zoom to a specific level, and zoom out. This allows you to grow or shrink the display. This lets the dispatcher get a view of a large layout in a single view or to zoom in on a specific control point as needed. This menu is also available in the generated CTC Panel program.

19.3.1.4 Panel menu

The `Panel` menu contains entries to add, edit, and delete panel elements (both track work and control) and also has an entry to edit the overall panel's configuration.

[6] These items are disabled when in "Simple Mode" or "OpenLCB Mode".

19.3.1.5 C/Mri menu

The `C/Mri` menu contains entries to add, edit, and delete C/MRI nodes on the C/MRI bus. The C/MRI nodes contain input and output ports that can be connected to things like occupancy detectors, turnout point state switches, signal LEDs (or lamps), and switch machine motors. They can also be connected to manual controls and indicators on control panels mounted over or beside the layout (eg "local" towers).

19.3.1.6 Azatrax menu

The `Azatrax` menu contains entries to add, edit, and delete Azatrax nodes on the USB bus. The Azatrax nodes contain sensors or control outputs that can be used to sense trains or operate signal LEDs (or lamps) or switch machine motors.

19.4 CTC Panel Code

The Dispatcher program creates Tcl scripts (programs). That is, each CTC Panel Window is implemented as a Tcl/Tk script file and in fact this is what is created when the window is "saved". The script file contains generated code, code that is created by the Dispatcher program (some of which is pre-written code that is copied to the script file). And some of the code is created by you the user of the program. [7] This code implements the CTC Panel that your model railroad's dispatcher will use to control some part of your model railroad during an operating session.

19.4.1 Wrapped CTC Panel Programs

The `Wrap As...` menu item on the file menu saves the CTC Panel Code as a StarPack, a self-contained executable program file that runs a Tcl/Tk program. This program can be run as-is, without needing any support files or code installed on the target system. You can create your panel on your desktop computer, which has the Model Railroad System installed and then you can "wrap" your panel program and then you can copy the generated executable program to a thumb drive and transfer the program to the computer used as your dispatcher's screen. The only 'gotcha' is that the computer used as your dispatcher's screen should be generally the same kind of computer as the desktop computer used to wrap the panel program – eg both should be 32-bit MS-Windows machines or both be 64-bit Linux machines, etc. You should also "save" your CTC Panel, if only to allow for future modifications and to document your CTC Panel.

19.4.2 Generated Code

The generated code consists of some prefix code including comments containing the panel's configuration, followed by code to load various packages used by the CTC Panel code, code to implement the panel itself, and code to initialize the C/MRI bus and initialize the nodes (boards) on the bus or code to initialize the Azatrax devices. Or code to connect to a LCC network, if the panel was created in OpenLCB mode.

[7] When running in "Simple Mode" or "OpenLCB mode" you won't be writing any code. All of the code will be generated by the Dispatcher program.

19.4.2.1 Configuring CTC Panel Windows

The configuration of CTC Panel Windows can be changed using the `Configure` entry of the `Panel` menu. This menu entry displays the Edit Panel Options Dialog, as shown below.

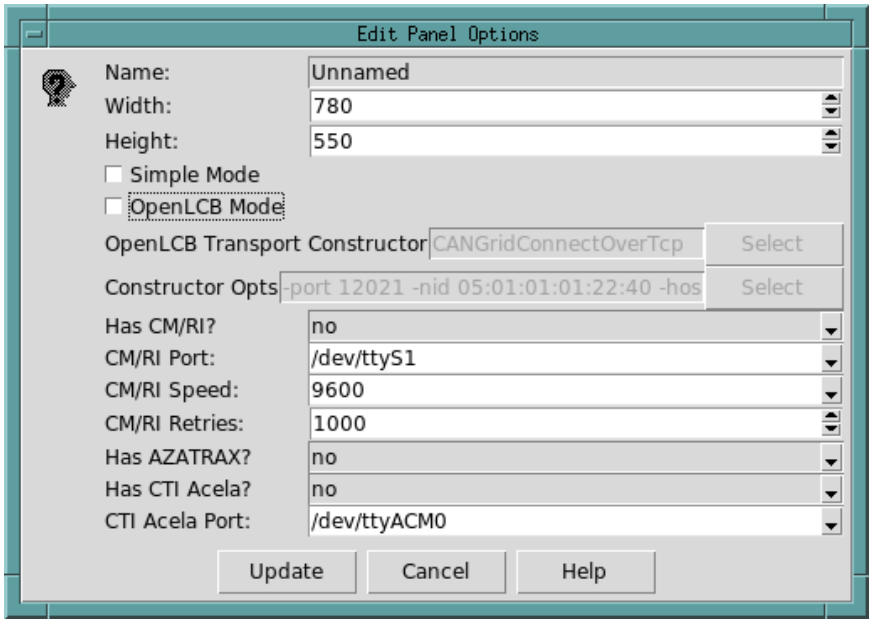


Figure 19.4 Edit Panel Options Dialog

This dialog box allows changing all of the same options as were set when the panel was created (see Section Creating a new CTC Panel).

19.4.2.2 Adding, Editing, and deleting elements to CTC Panel Windows

CTC Panel elements can be added, edited, or deleted with the `Add`, `Edit`, and `Delete` entries of the `Panel` menu. There are twenty element types to select from. CTC Panel track work elements can also be added directly from the Track work Node Graphs using the right button node menu. Every CTC Panel element has a unique name, is part of a control point, [8] and has an X, Y location on either the track work schematic (for track work elements) or the control area (for control elements). The X, Y location(s) can be either set by entering the coordinates directly (this allows precise positioning) or by using cross hairs to position elements using the pointer device (eg mouse). [9] Additionally, element specific options are available for each element. When entering Switch Plates in "Simple Mode", there is a provision for entering the type, serial number, sub-elements of the Azatrax device and the name of the track work switch. All of the command script entries are disabled, although the generated scriptlets are shown (for the curious).

Additionally, when in "OpenLCB mode", instead of scripts, various elements will have entries for LCC event ids instead. LCC event ids are 64-bit numbers, represented as 8 pairs of hexadecimal digits separated by periods. For track work, there are two LCC event ids used for occupancy, one for the occupied event (a train enters the block) and one for the

[8] For mainline trackage a control point of “Main” can be used.
[9] After placing a device with the cross hairs, it is possible to adjust the coordinates for added precision.

not occupied event (a train leaves the block). For turnouts there are a pair of event ids for the point position sensor: one for the normal position and one for the reverse position. For control elements, there are event ids to be produced for lever positions or button pushes and event ids that will be consumed to update the indications. And for signals there is a list of aspects: the list of colors (top to bottom) and the event id to be consumed to display that aspect. See section Using the Dispatcher program with layouts designed in XtrackCAD for specific use with XtrackCAD.

19.4.2.3 Adding, Editing, and deleting C/Mri nodes to CTC Pane Windows

C/Mri nodes can be added, edited, or deleted with the `Add`, `Edit`, and `Delete` entries of the `C/Mri` menu. Each node has a unique name and unique UA (address). There are three supported board types: SMINI (Super Mini), USIC (Universal Serial Interface Card) and SUSIC (Super Universal Serial Interface Card).

19.4.2.4 Adding, Editing, and deleting Azatrax nodes to CTC Panel Windows

Azatrax nodes can be added, edited, or deleted with the `Add`, `Edit`, and `Delete` entries of the `Azatrax` menu. Each node has a unique name and unique serial number.

19.4.3 User Code

The user code is editable with the `User Code` entry of the `Edit` menu. This menu entry either starts a simple edit window or starts a user-specified external editor (See Section Configurable Options). In addition to directly editing the user code, one or more pre-written modules can be inserted and a skeleton main loop can be created. When using the program in "Simple Mode" the user code menu items are disabled. All code is generated by the Dispatcher program. The functions and logic are limited to the canned "Simple Mode" functionality. It is possible to later turn off "Simple Mode" in the panel's configuration (see Section Configuring CTC Panel Windows). When using the program in "OpenLCB mode" the user code menu items are also disabled. All code is generated by the Dispatcher program. It is presumed that any special logic needed will be handled by a logic node on the LCC network (such as the OpenLCB_Logic daemon or logic blocks in a Tower-LCC node).

19.4.3.1 Insert-able Modules

These are a collection of SNIT types, in name spaces, that encapsulate various common types of things that a CTC Panel implements, including blocks, turnouts, signals, signal plates, control points, and radio groups (commonly used to implement a software track selection matrix for a yard or terminal). See Insertable Module Library for details of these code modules.

19.4.3.2 The Main Loop

A skeleton main loop can be generated, but you will need to modify it to implement the actual logic of your CTC Panels. The basic main loop is an endless, "real time" loop, that reads in all of the input ports, invokes all of the track work, and then writes all of the output ports. It is necessary to decode the input bytes into bit fields which can be stored in various types. The occupation state and switch point state information sensed from the inputs is used, along with the settings of the control elements (switch plates, signal plates, etc.) is tested and logical tests are applied to determine things like signal aspects and switch motor values, etc. These values are then packed into the vectors (lists) of output bytes which are then written to the output ports.

19.5 Add CMRI Node Dialog

This dialog box adds a C/MRI node to the C/MRI bus. These nodes are the SUSIC, USIC, SMINI boards. Each board has a name and an address (0 to 127). If the board is a SMINI board, it can have a count of yellow signals and a yellow signal map and for SUSIC and USIC nodes, it has a count on input and output ports and a map of card types. There is also a delay value. [10] The name will be a SNIT object instance name and should start with a letter and contain only letters, digits, period, underscore, or dash.

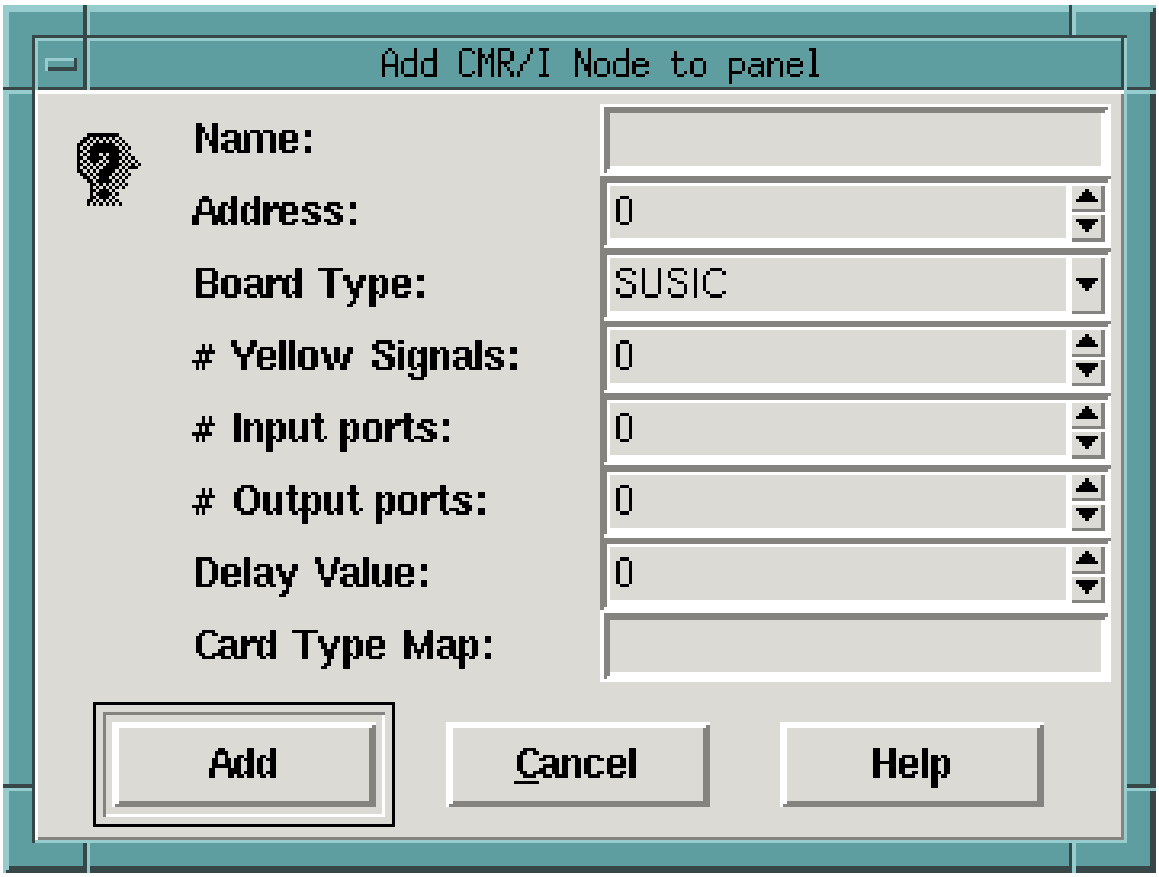


Figure 19.5 Add / Edit CMR/I Node Dialog

19.6 Select CMRI Node Dialog

This dialog box selects an existing C/MRI node. It is possible to specify a pattern to narrow the list of results.

[10] Only meaningful for the older USIC boards.

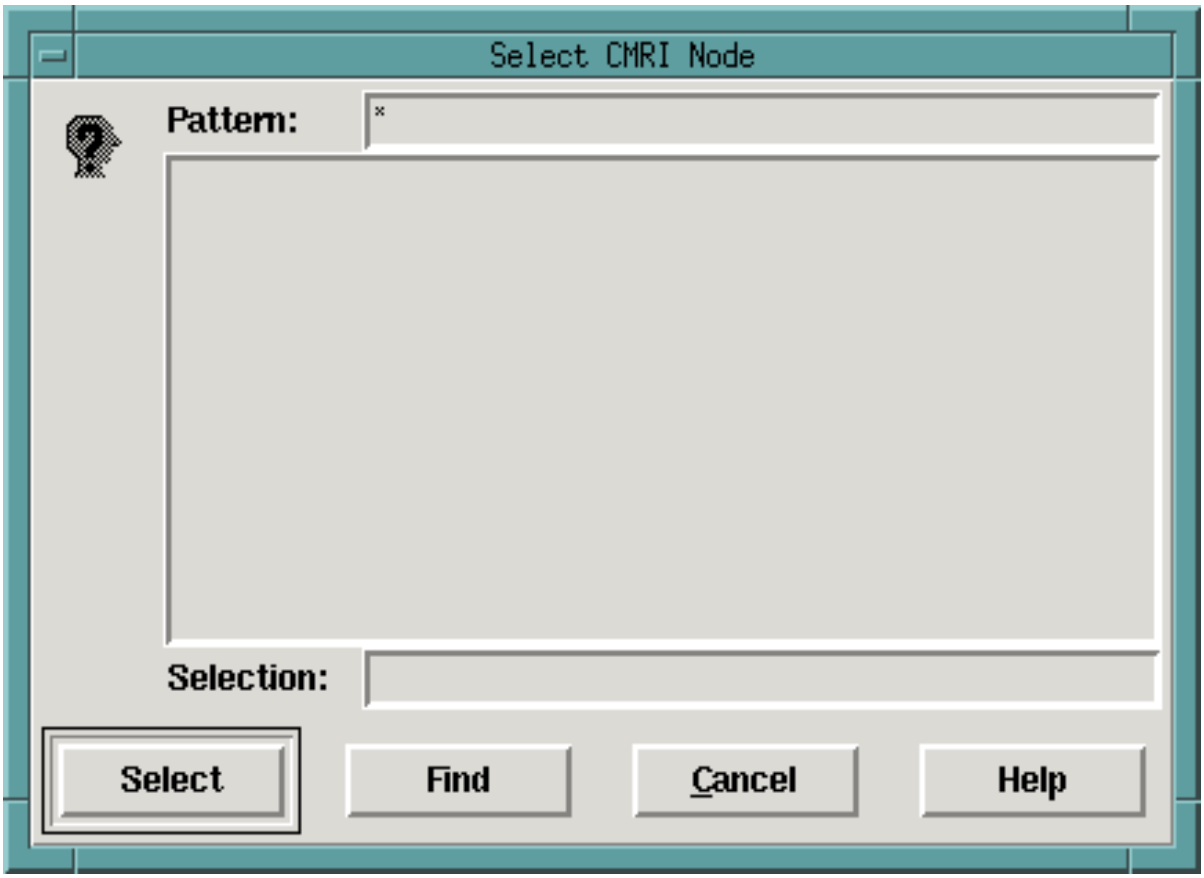


Figure 19.6 Select CMR/I Node Dialog

19.7 Add Azatrax Node Dialog

This dialog box adds an Axatrax device and gives it a name that can be used with the user code to access the device's state information and to actuate its channels. The dialog box asks for a name and the device's type and serial number.

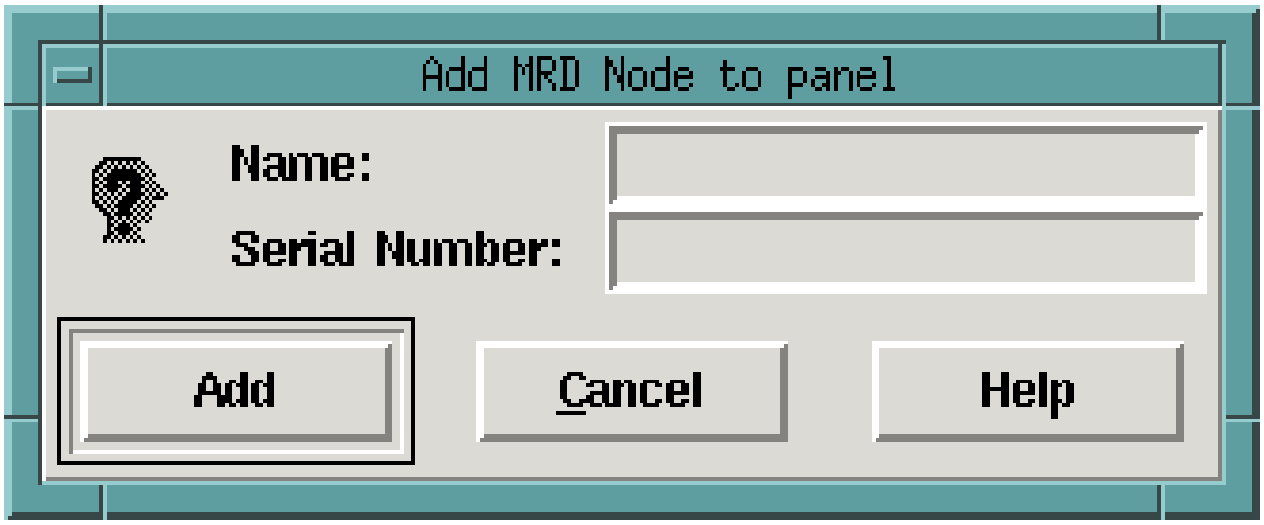


Figure 19.7 Add / Edit Axatrax Node Dialog

19.8 Select Azatrax Node Dialog

This dialog box selects an existing Axatrax device. It is possible to specify a pattern to narrow the list of results.

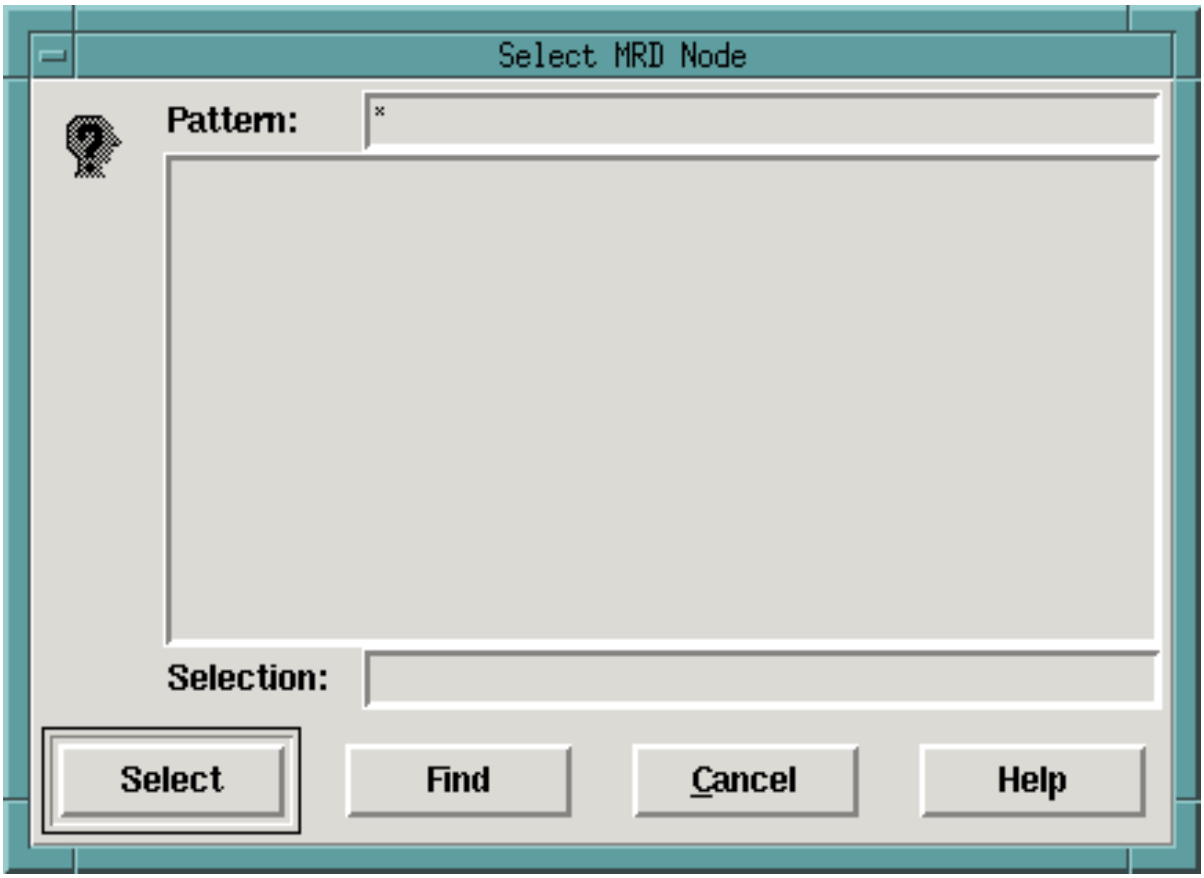


Figure 19.8 Select Axatrax Node Dialog

19.9 Add Panel Object Dialog

This dialog box adds an object to either the schematic (track work) panel or control panel. Each object is of a specified type and has a unique name, is part of a control point, and has various attributes, such as a location (X and Y coordinates), orientation, label, etc.

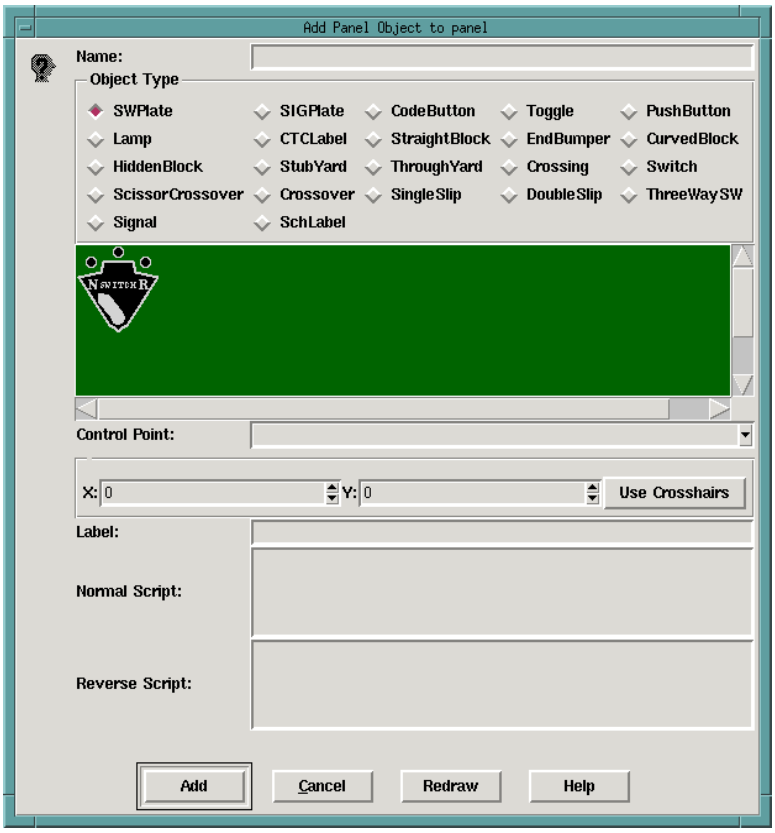


Figure 19.9 Add / Edit Panel Object Dialog

19.10 Select Panel Object Dialog

This dialog box selects an existing object on the schematic (track work) panel or control panel.It is possible to specify a pattern to narrow the list of results.

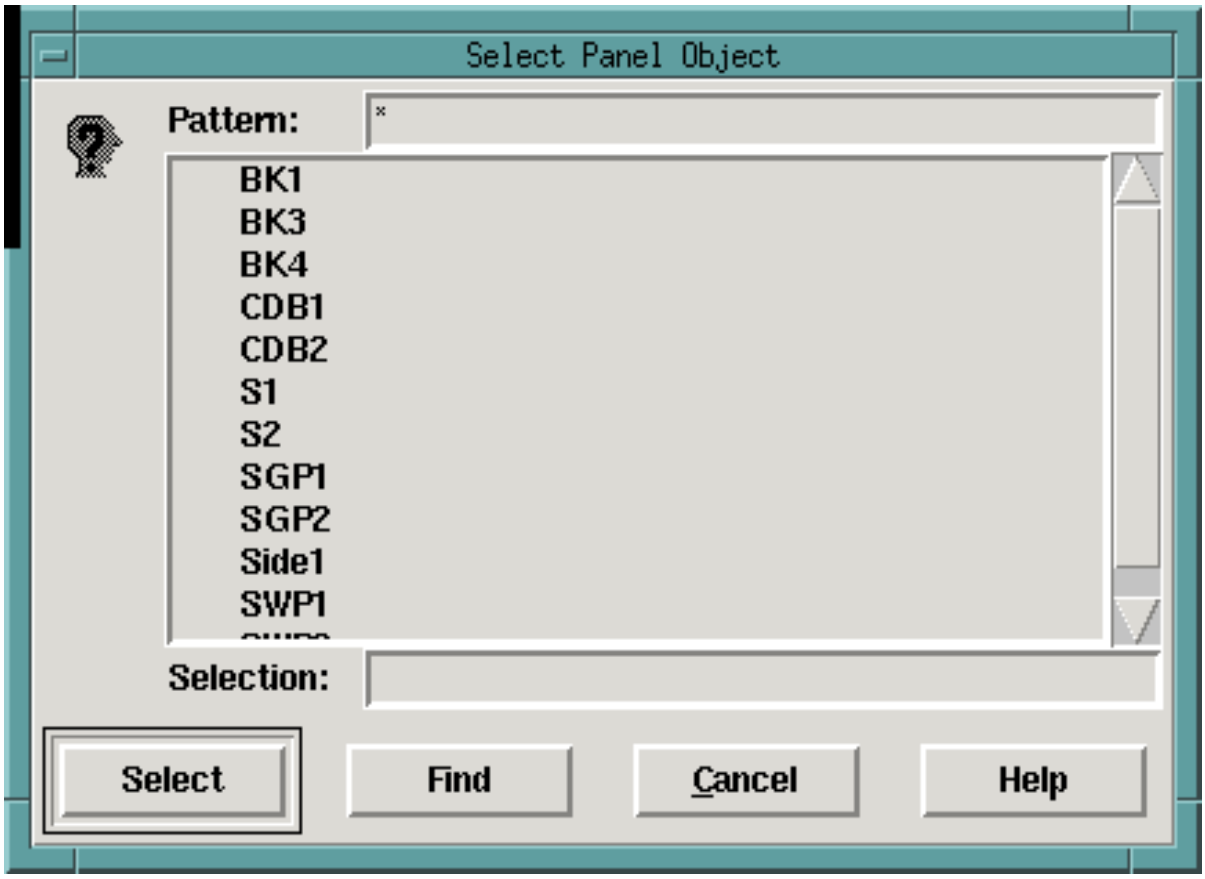


Figure 19.10 Select Panel Object Dialog

19.11 Edit User Code Dialog

This dialog box displays the user code and provides a simple text editor to edit the user code.

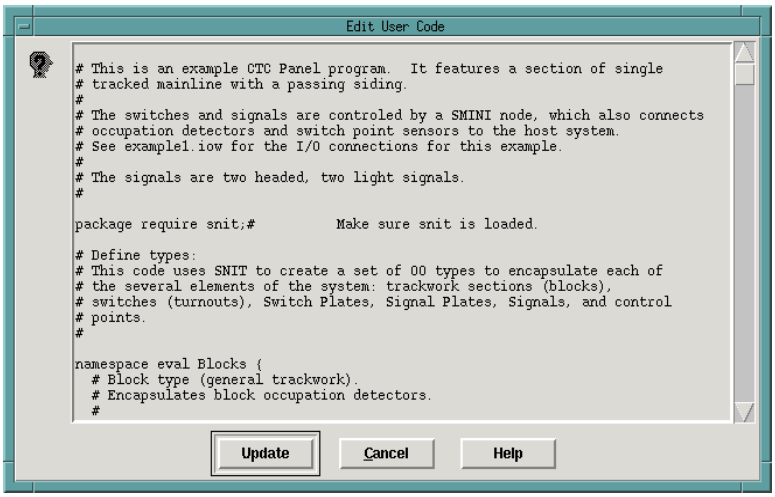


Figure 19.11 Edit User Code Dialog

19.12 Find Node Dialog

This dialog box is used to find nodes by number in the node graph.

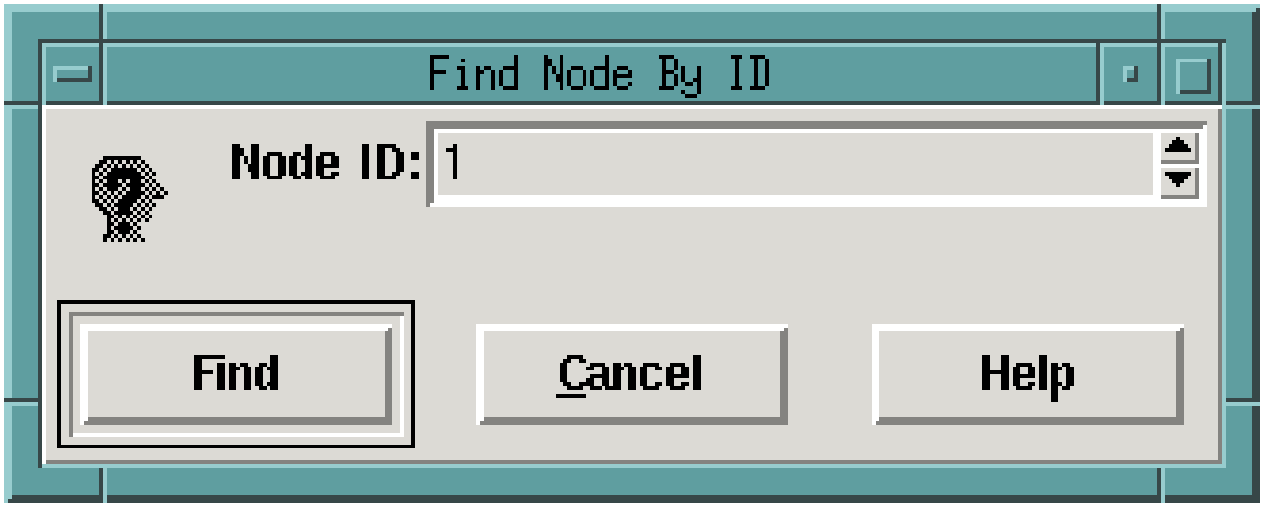


Figure 19.12 Find Node Dialog

19.13 Print Dialog

This dialog box selects the output PDF file and paper size for the print operations. [11]

[11] Really it is a save to PDF file. To really print you need to open the PDF file with a PDF viewer and then select the Print function of the viewer to then print the file.

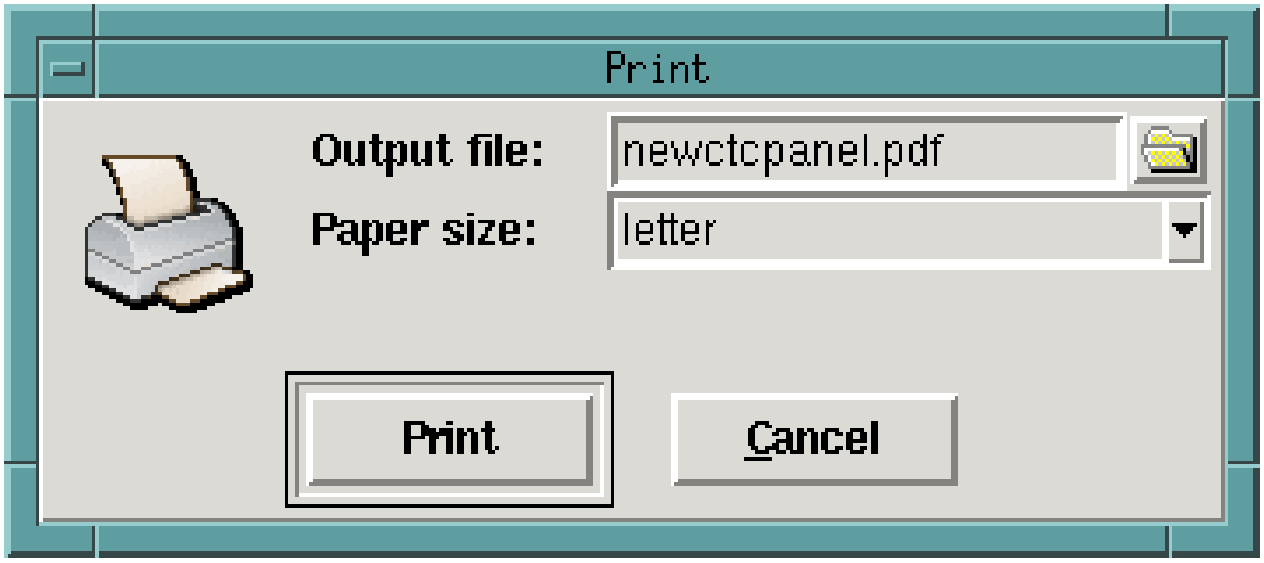


Figure 19.13 Print Dialog

19.14 Select Panel Dialog

This dialog box selects the panel to add track work from the node graph to.

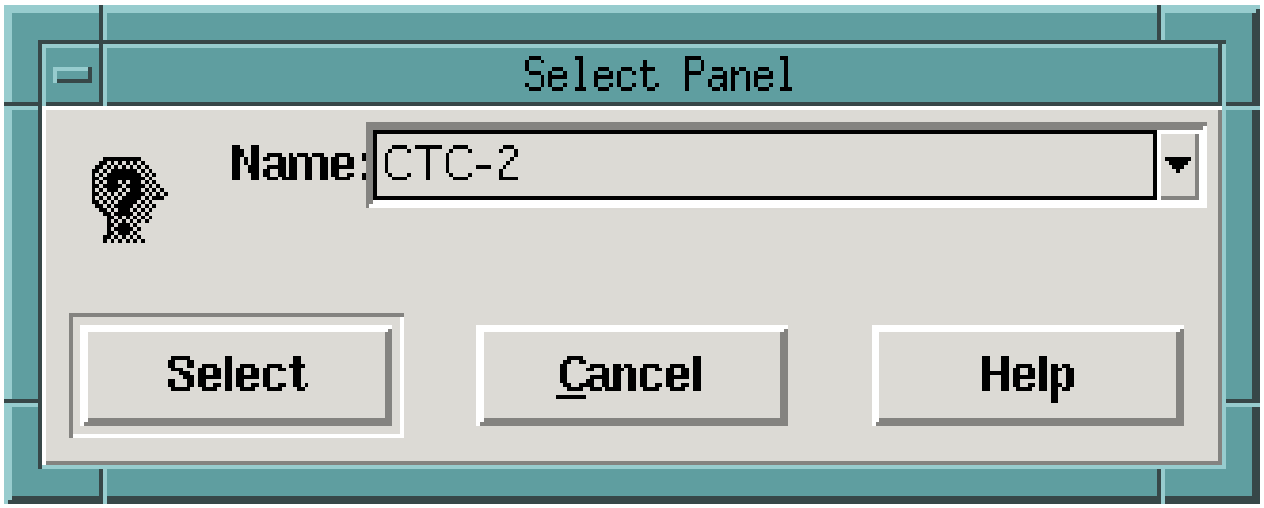


Figure 19.14 Select CTC Panel Dialog

19.15 Using the Dispatcher program with layouts designed in XtrackCAD

XtrackCAD includes a feature called "Layout Control Elements", where the layout designer can include information for the layout control software (eg The Model Railroad System) in the layout file. The Dispatcher includes a parser for XtrackCAD files and can extract this information and copy it into a CTC Panel, if it is formatted properly. The specific elements that the Dispatcher program can access include blocks (for occupancy detection), switch motors (for turnout control), and signals for signal aspect display.

19.15.1 LCC event id format.

A LCC event id is a 64-bit number, represented as eight pairs of hexadecimal digits (0-9, a-f/A-F) separated by periods (.). Each pair represents one 8-bit byte of the event id. This event id is either produced by a sensor or logic element or is consumed by a control/device or a logic element.

19.15.2 XTrackCAD "script" formats.

For blocks the occupancy script contains a pair of LCC event ids, separated by a colon (:). The first LCC event id is produced by the occupancy detector when the train enters the block and the second LCC event id is produced by the occupancy detector when the train leaves the block.

For switch motors the point sense script contains a pair of LCC event ids, separated by a colon (:). The first LCC event id is produced by the point sensor when the points are aligned in the "normal" position (typically aligned to the main) and the second LCC event id is produced by the point sensor when the points are aligned in the "reverse" position (typically aligned to the spur). The normal and reverse script each contain a single LCC event id. These events are produced by the CTC Panel when the control point Code button is pressed (clicked) and are consumed by the switch motor.

For signals, the aspect name is a space separated list of the color(s) of the signal heads from top to bottom and the aspect script is a LCC event id that is consumed to produce that aspect. Presumably, the LCC event id is produced by a logic element (presumably a mast group in a Tower-LCC or similar device) or virtual track circuit in a Tower-LCC or similar device.

19.15.3 Layout Controls Dialog

When an XTrackCAD has been loaded, the `View` menu item `Layout Controls` becomes enabled and can be used to display all of the layout control elements loaded from the layout file. These controls can be viewed or extracted to CSV files (suitable for importing into Excel or oocalc).

19.16 Insertable Module Library

19.16.1 Track Work type

These are types related to track work.

There are two types defined:

19.16.1.1 Blocks::Block

This type defines two methods:

```
occupiedp {}
setoccupied {value}
```

The `occupiedp` method return a boolean value depending on the state of the block (occupied or not). The `setoccupied` sets the state of the block (as written, a value of 1 means occupied and a value of 0 means not occupied).

19.16.1.2 Switches::Switch

This type defines the same methods as Block, plus these four additional methods:

```
getstate {}
setstate {statebits}
motorbits {}
setmotor {mv}
```

The `getstate` and `setstate` methods relate to the state of the points.

The `motorbits` and `setmotor` methods handle the switch motor.

19.16.2 Switch Plate type

This defines one type, `SwitchPlates::SwitchPlate`.

Its constructor takes one additional argument, typically an instance of a Switches::Switch, to which it delegates methods to. In addition, it adds two methods:

```
setlever {pos}
getlever {}
```

These two methods relate to the switch plate's lever position.

19.16.3 Signal types

There are three signal modules: two LEDs per signal head (red, green), three LEDs per signal head (red, yellow, green), and single bi-color LED per head.

All of the signal types take one option `-signal` which is a signal type panel object and they define two methods:

```
setaspect {a}
getaspect {}
```

They vary only in the aspect codes and the aspect bits defined.

The types defined are:

- Two Aspect Color Light
  - Signals::OneHead
  - Signals::TwoHead
- Three Aspect Color Light
  - Signals::OneHead
  - Signals::TwoHead
  - Signals::ThreeHead
- Three Aspect Search Light
  - Signals::OneHead
  - Signals::TwoHead
  - Signals::ThreeHead

19.16.4 Signal Plate type

This defines one type, `SignalPlates::SignalPlate`, which takes one option `-signalplate`, which is the name of the CTC panel Signal Plate. It defines these methods:

```
setlever {pos}
getlever {}
setdot {dir}
```

The `setlever` and `getlever` methods store and fetch the lever state. The @ setdot method update the indicator lamps on the signal plate.

19.16.5 Control Point type

This module defines one type, `*`c ControlPoints::ControlPoint, which takes one option, `-cpname`, which is the name of as control point. It defines one method, `code`, which takes no arguments and would typically be bound to a code button. This method invokes all of the switch plates and signal plates in the named control point.

19.16.6 Radio Group type

This module defines one type, `Groups::Group`, which takes one option `-buttonmap`, which is an even element list containing button names and values, with the odd elements being the button names and the even elements the values. It defines two methods:

```
getvalue {}
setvalue {newvalue}
```

The `setvalue` method would be bound to a button to set that buttons value. The `getvalue` method would be called to fetch the set value.

Chapter 20

Dispatcher Examples

These are four examples created using the Dispatcher program. The code files are included and can be used as references or even modified to suit some part of your layout.

20.1 Example 1: Simple siding on single track mainline

This example, shown below, implements a simple passing siding on a single track main line. There are two control points, one at each end of the siding. Both control points are handled with a single SMINI board.

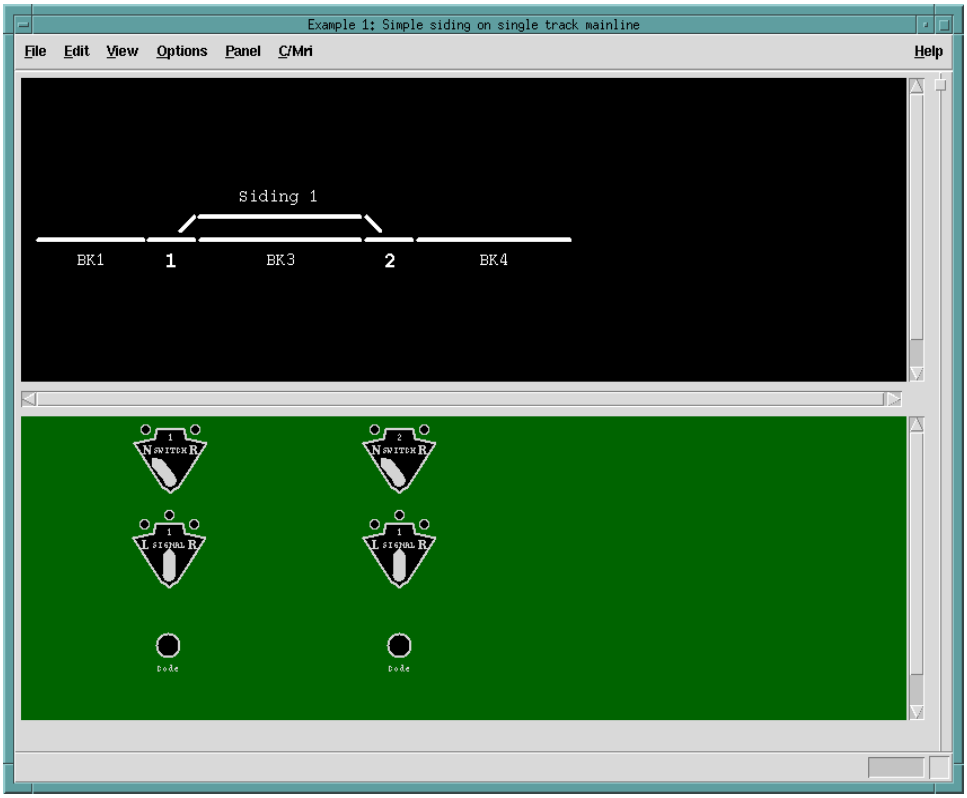


Figure 20.1 Example 1: Simple siding on single track mainline

Here is the code:

```
# Add User code after this line
# This is an example CTC Panel program.  It features a section of single
# tracked mainline with a passing siding.
#
# The switches and signals are controled by a SMINI node, which also connects
# occupation detectors and switch point sensors to the host system.
# See example1.iow for the I/O connections for this example.
#
# The signals are two headed, two light signals.
#
package require snit;#          Make sure snit is loaded.
# Define types:
# This code uses SNIT to create a set of OO types to encapsulate each of
# the several elements of the system: trackwork sections (blocks),
# switches (turnouts), Switch Plates, Signal Plates, Signals, and control
# points.
#
namespace eval Blocks {
  # Block type (general trackwork).
  # Encapsulates block occupation detectors.
  #
  snit::type Block {
    # Occupation state values
    typevariable OCC 1
    typevariable CLR 0
    # Occupation state bit
    variable occupiedbit
    constructor {} {
      set occupiedbit $CLR;#    Initialize to clear.
    }
    # Occupation state methods
    method occupiedp {} {return [expr {$occupiedbit == $OCC}]}
    method setoccupied {value} {
      set occupiedbit $value
    }
  }
  Block BK1;#   Block 1
  Block SD1;#   Siding
  Block BK3;#   Block 3
  Block BK4;#   Block 4
}
namespace eval Switches {
  # Switch type (turnout)
  # Encapsulates a switch (turnout), including its OS (delegated to a Block
  # object), its switch motor, and its point position sensor (its state).
  snit::type Switch {
    component block;#                   OS section
    delegate method * to block;#        Delegate block methods
    variable state unknown;#            Sense state (point position)
    # Motor bit values
    typevariable NOR 1;# 01
    typevariable REV 2;# 10
    variable motor;#                    Motor bits -- used to drive switch
#                                       motor.
    constructor {} {
      #                                 Install OS section
      install block using Blocks::Block %AUTO%
      # Initialize motor bits
      set motor $NOR
    }
    # State methods
    method getstate {} {return $state}
    method setstate {statebits} {
      if {$statebits == $NOR} {
        set state normal
      } elseif {$statebits == $REV} {
        set state reverse
      } else {
        set state unknown
      }
    }
    # Motor bit methods
    method motorbits {} {return $motor}
    method setmotor {mv} {
      switch -exact $mv {
        normal {set motor $NOR}
        reverse {set motor $REV}
      }
    }
  }
```

```
  Switch S1;#           Switch 1
  Switch S2;#           Switch 2
}
namespace eval SwitchPlates {
  # Switch Plate
  # Encapsulates a switch plate, implementing its lever position.
  snit::type SwitchPlate {
    component switch
    delegate method * to switch
    variable leverpos unknown
    constructor {sw} {
      set switch $sw
    }
    method setlever {pos} {set leverpos $pos}
    method getlever {} {return $leverpos}
  }
  SwitchPlate SWP1 Switches::S1
  SwitchPlate SWP2 Switches::S2
}
namespace eval Signals {
  # Signal types.  Encapsulates a signal's aspect.
  snit::type OneHead {
    # Single head signals have three states: dark, green or red.
    typevariable aspects -array {
      Dark      0x00
      Green     0x01
      Red       0x02
    }
    variable aspectbits
    constructor {} {
      set aspectbits $aspects(Dark)
    }
    method setaspect {a} {set aspectbits $aspects($a)}
    method getaspect {}  {return $aspectbits}
  }
  snit::type TwoHead {
    # Two head signals have four states: dard, green over red, red over green,
    # and red over red.
    typevariable aspects -array {
      Dark      0x00
      GreenRed  0x06
      RedGreen  0x09
      RedRed    0x0A
    }
    variable aspectbits
    constructor {} {
      set aspectbits $aspects(Dark)
    }
    method setaspect {a} {set aspectbits $aspects($a)}
    method getaspect {}  {return $aspectbits}
  }
  TwoHead CP1E;#        Heading into switch 1 from the west (block 1)
  TwoHead CP1WM;#       Heading into switch 1 from the east on the main
  TwoHead CP1WS;#       Heading into switch 1 from the east from the siding
  TwoHead CP2EM;#       Heading into switch 2 from the west on the main
  TwoHead CP2ES;#       Heading into switch 2 from the west from the siding
  TwoHead CP2W;#        Heading into switch 2 from the east (block 4)
}
namespace eval SignalPlates {
  # Signal Plate, encapsulating a signal plate with its lever and indicators.
  snit::type SignalPlate {
    variable leverpos unknown
    option -signalplate -default {} -readonly yes
    constructor {args} {
      $self configurelist $args
    }
    method setlever {pos} {set leverpos $pos}
    method getlever {} {return $leverpos}
    method setdot {dir} {
      switch $dir {
        left {
          MainWindow ctcpanel seti $options(-signalplate) L on
          MainWindow ctcpanel seti $options(-signalplate) C off
          MainWindow ctcpanel seti $options(-signalplate) R off
        }
        right {
          MainWindow ctcpanel seti $options(-signalplate) L off
          MainWindow ctcpanel seti $options(-signalplate) C off
          MainWindow ctcpanel seti $options(-signalplate) R on
        }
        none -
```

```
        default {
          MainWindow ctcpanel seti $options(-signalplate) L off
          MainWindow ctcpanel seti $options(-signalplate) C on
          MainWindow ctcpanel seti $options(-signalplate) R off
        }
      }
    }
  }
  SignalPlate SGP1 -signalplate SGP1;#  Signal plate for control point 1
  SignalPlate SGP2 -signalplate SGP2;#  Signal plate for control point 2
}
namespace eval ControlPoints {
  # Control points.  Used to implement code buttons.
  # Encapsulates a control point
  snit::type ControlPoint {
    option -cpname -readonly yes -default {}
    constructor {args} {
      $self configurelist $args
    }
    method code {} {
      foreach swp [MainWindow ctcpanel objectlist $options(-cpname) SwitchPlates] {
        MainWindow ctcpanel invoke $swp
      }
      foreach sgp [MainWindow ctcpanel objectlist $options(-cpname) SignalPlates] {
        MainWindow ctcpanel invoke $sgp
      }
    }
  }
  ControlPoint CP1 -cpname CP1;#        Control point 1
  ControlPoint CP2 -cpname CP2;#        Control point 2
}
# Main Loop Start
while {true} {
  # Read all ports
  set CP1_2_inbits [CP1_2 inputs]
  # Occupation Detectors: (Input Port A)
  set tempByte [lindex $CP1_2_inbits 0]
  Blocks::BK1  setoccupied [expr {$tempByte      & 0x01}]
  Switches::S1 setoccupied [expr {$tempByte » 1 & 0x01}]
  Blocks::BK3  setoccupied [expr {$tempByte » 2 & 0x01}]
  Blocks::SD1  setoccupied [expr {$tempByte » 3 & 0x01}]
  Switches::S2 setoccupied [expr {$tempByte » 4 & 0x01}]
  Blocks::BK4  setoccupied [expr {$tempByte » 5 & 0x01}]
  # Switch point state switches (Input Port B)
  set tempByte [lindex $CP1_2_inbits 1]
  Switches::S1 setstate [expr {$tempByte      & 0x03}]
  Switches::S2 setstate [expr {$tempByte » 2 & 0x03}]
  # Invoke all trackwork and get occupicency
  MainWindow ctcpanel invoke Side1
  MainWindow ctcpanel invoke BK1
  MainWindow ctcpanel invoke BK3
  MainWindow ctcpanel invoke BK4
  # Initialize all signals to Red
  Signals::CP1E setaspect RedRed
  Signals::CP1WM setaspect RedRed
  Signals::CP1WS setaspect RedRed
  Signals::CP2EM setaspect RedRed
  Signals::CP2ES setaspect RedRed
  Signals::CP2W setaspect RedRed
  set dot1 none;# Assume no direction of travel through control point 1
  if {![MainWindow ctcpanel invoke S1]} {
    # Switch1 OS is clear.  We can start to move the points
    Switches::S1 setmotor [SwitchPlates::SWP1 getlever]
    # get current point state
    switch [Switches::S1 getstate] {
      normal {;# Aligned for the Main.  Set the mainline signals.
        if {![Blocks::BK1 occupiedp] &&
            [string equal [SignalPlates::SGP1 getlever] "left"]} {
          # Clear to left
          Signals::CP1WM setaspect GreenRed
          set dot1 left
        }
        if {![Blocks::BK3 occupiedp] &&
            [string equal [SignalPlates::SGP1 getlever] "right"]} {
          # Clear to right
          Signals::CP1E setaspect GreenRed
          set dot1 right
        }
        # Set plate indicators
        MainWindow ctcpanel seti SWP1 N on
        MainWindow ctcpanel seti SWP1 R off
```

```
    }
    reverse {;# Aligned for the siding.  Set siding signals
      if {![Blocks::BK1 occupiedp] &&
          [string equal [SignalPlates::SGP1 getlever] "left"]} {
        # Clear to left
        Signals::CP1WS setaspect RedGreen
        set dot1 left
      }
      if {![Blocks::SD1 occupiedp] &&
          [string equal [SignalPlates::SGP1 getlever] "right"]} {
        # Clear to right
        Signals::CP1E setaspect RedGreen
        set dot1 right
      }
      # Set plate indicators
      MainWindow ctcpanel seti SWP1 R on
      MainWindow ctcpanel seti SWP1 N off
    }
    default {;# Points still moving.
      # Set plate indicators
      MainWindow ctcpanel seti SWP1 R off
      MainWindow ctcpanel seti SWP1 N off
    }
  }
}
# Set DOT on switch plate
SignalPlates::SGP1 setdot $dot1
# Switch 2 is much the same.
set dot2 none
if {![MainWindow ctcpanel invoke S2]} {
  Switches::S2 setmotor [SwitchPlates::SWP2 getlever]
  switch [Switches::S2 getstate] {
    normal {
      if {![Blocks::BK4 occupiedp] &&
          [string equal [SignalPlates::SGP2 getlever] "right"]} {
        Signals::CP2EM setaspect GreenRed
        set dot2 right
      }
      if {![Blocks::BK3 occupiedp] &&
          [string equal [SignalPlates::SGP2 getlever] "left"]} {
        Signals::CP2W setaspect GreenRed
        set dot2 left
      }
      MainWindow ctcpanel seti SWP2 N on
      MainWindow ctcpanel seti SWP2 R off
    }
    reverse {
      if {![Blocks::BK4 occupiedp] &&
          [string equal [SignalPlates::SGP2 getlever] "right"]} {
        Signals::CP2ES setaspect RedGreen
        set dot2 right
      }
      if {![Blocks::SD1 occupiedp] &&
          [string equal [SignalPlates::SGP2 getlever] "left"]} {
        Signals::CP2W setaspect RedGreen
        set dot2 left
      }
      MainWindow ctcpanel seti SWP2 R on
      MainWindow ctcpanel seti SWP2 N off
    }
    default {
      MainWindow ctcpanel seti SWP2 R off
      MainWindow ctcpanel seti SWP2 N off
    }
  }
}
SignalPlates::SGP2 setdot $dot2
# Approach lighting -- darken signals facing empty blocks.
if {![Blocks::BK1 occupiedp]} {
  Signals::CP1E setaspect Dark
}
if {![Blocks::BK3 occupiedp]} {
  Signals::CP1WM setaspect Dark
  Signals::CP2EM setaspect Dark
}
if {![Blocks::SD1 occupiedp]} {
  Signals::CP1WS setaspect Dark
  Signals::CP2ES setaspect Dark
}
if {![Blocks::BK4 occupiedp]} {
  Signals::CP2W setaspect Dark
```

```
  }
  # Pack output bits
  # Output Port A Card 0(CP1E and CP1WM)
  set CP1_2_outbits [expr {[Signals::CP1E  getaspect] | \
                           [Signals::CP1WM getaspect] « 4}]
  # Output Port B Card 0 (CP1WS and S1)
  lappend CP1_2_outbits [expr {[Signals::CP1WS getaspect] | \
                               [Switches::S1   motorbits] « 4}]
  # Output Port C Card 0 (CP2EM and CP2ES)
  lappend CP1_2_outbits [expr {[Signals::CP2EM getaspect] | \
                               [Signals::CP2ES getaspect] « 4}]
  # Output Port A Card 2 (CP2W and S2)
  lappend CP1_2_outbits [expr {[Signals::CP2W getaspect] | \
                               [Switches::S2  motorbits] « 4}]
  lappend CP1_2_outbits 0 0;# Output Ports B and C Card 2are not used.
#  puts stderr "*** CP1_2_outbits = $CP1_2_outbits"
  # Write all output ports
  eval [list CP1_2 outputs] $CP1_2_outbits
  update;# Update display
}
# Main Loop End
```

And the I/O Worksheet:

```
SMINI @ UA 0:
Card No. 0 Output
-----------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED               |
-----------------------------------------------------------------
|    | 0 | 1 |Lower Green \                                   |
|    | 1 | 2 |Lower Red   | CP1E Signal                       |
|    | 2 | 3 |Upper Green |                                   |
| A  | 3 | 4 |Upper Red   /                                   |
|    | 4 | 5 |Lower Green \                                   |
|    | 5 | 6 |Lower Red   | CP1WM Signal                      |
|    | 6 | 7 |Upper Green |                                   |
|    | 7 | 8 |Upper Red   /                                   |
-----------------------------------------------------------------
|    | 0 | 9 |Lower Green \                                   |
|    | 1 |10 |Lower Red   | CP1WS Signal                      |
|    | 2 |11 |Upper Green |                                   |
| B  | 3 |12 |Upper Red   /                                   |
|    | 4 |13 |Normal \ Switch 1                               |
|    | 5 |14 |Reverse/ Motor                                  |
|    | 6 |15 |                                                |
|    | 7 |16 |                                                |
-----------------------------------------------------------------
|    | 0 |17 |Lower Green \                                   |
|    | 1 |18 |Lower Red   | CP1E Signal                       |
|    | 2 |19 |Upper Green |                                   |
| C  | 3 |20 |Upper Red   /                                   |
|    | 4 |21 |Lower Green \                                   |
|    | 5 |22 |Lower Red   | CP1WM Signal                      |
|    | 6 |23 |Upper Green |                                   |
|    | 7 |24 |Upper Red   /                                   |
-----------------------------------------------------------------
|    | 0 |25 |                                                |
|    | 1 |26 |                                                |
|    | 2 |27 |                                                |
| D  | 3 |28 |                                                |
|    | 4 |29 |                                                |
|    | 5 |30 |                                                |
|    | 6 |31 |                                                |
|    | 7 |32 |                                                |
-----------------------------------------------------------------
Card No. 1 Input
-----------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED               |
-----------------------------------------------------------------
|    | 0 | 1 | Block 1 Occupation Detector                    |
|    | 1 | 2 | Switch 1 Occupation Detector                   |
|    | 2 | 3 | Block 3 Occupation Detector                    |
| A  | 3 | 4 | Siding Occupation Detector                     |
|    | 4 | 5 | Switch 2 Occupation Detector                   |
|    | 5 | 6 | Block 4 Occupation Detector                    |
|    | 6 | 7 |                                                |
|    | 7 | 8 |                                                |
-----------------------------------------------------------------
|    | 0 | 9 | Switch 1 state (normal)                        |
|    | 1 |10 | Switch 1 state (reverse)                       |
|    | 2 |11 | Switch 2 state (normal)                        |
```

```
| B  | 3 |12 | Switch 2 state (reverse)                        |
|    | 4 |13 |                                                 |
|    | 5 |14 |                                                 |
|    | 6 |15 |                                                 |
|    | 7 |16 |                                                 |
-------------------------------------------------------------
|    | 0 |17 |                                                 |
|    | 1 |18 |                                                 |
|    | 2 |19 |                                                 |
| C  | 3 |20 |                                                 |
|    | 4 |21 |                                                 |
|    | 5 |22 |                                                 |
|    | 6 |23 |                                                 |
|    | 7 |24 |                                                 |
-------------------------------------------------------------
|    | 0 |25 |                                                 |
|    | 1 |26 |                                                 |
|    | 2 |27 |                                                 |
| D  | 3 |28 |                                                 |
|    | 4 |29 |                                                 |
|    | 5 |30 |                                                 |
|    | 6 |31 |                                                 |
|    | 7 |32 |                                                 |
-------------------------------------------------------------
Card No. 2 Output
-------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED             |
-------------------------------------------------------------
|    | 0 | 1 |Lower Green \                                    |
|    | 1 | 2 |Lower Red   | CP2W Signal                        |
|    | 2 | 3 |Upper Green |                                    |
| A  | 3 | 4 |Upper Red   /                                    |
|    | 4 | 5 |Normal \ Switch 2                                |
|    | 5 | 6 |Reverse/ Motor                                   |
|    | 6 | 7 |                                                 |
|    | 7 | 8 |                                                 |
-------------------------------------------------------------
|    | 0 | 9 |                                                 |
|    | 1 |10 |                                                 |
|    | 2 |11 |                                                 |
| B  | 3 |12 |                                                 |
|    | 4 |13 |                                                 |
|    | 5 |14 |                                                 |
|    | 6 |15 |                                                 |
|    | 7 |16 |                                                 |
-------------------------------------------------------------
|    | 0 |17 |                                                 |
|    | 1 |18 |                                                 |
|    | 2 |19 |                                                 |
| C  | 3 |20 |                                                 |
|    | 4 |21 |                                                 |
|    | 5 |22 |                                                 |
|    | 6 |23 |                                                 |
|    | 7 |24 |                                                 |
-------------------------------------------------------------
|    | 0 |25 |                                                 |
|    | 1 |26 |                                                 |
|    | 2 |27 |                                                 |
| D  | 3 |28 |                                                 |
|    | 4 |29 |                                                 |
|    | 5 |30 |                                                 |
|    | 6 |31 |                                                 |
|    | 7 |32 |                                                 |
-------------------------------------------------------------
EOF
```

20.2 Example 2: Mainline with an industrial siding

This example, shown below, implements an industrial siding on a single track main line. There are two control points, one at each end of the siding. This example uses three SMINI boards, one for each control point and one for the siding.

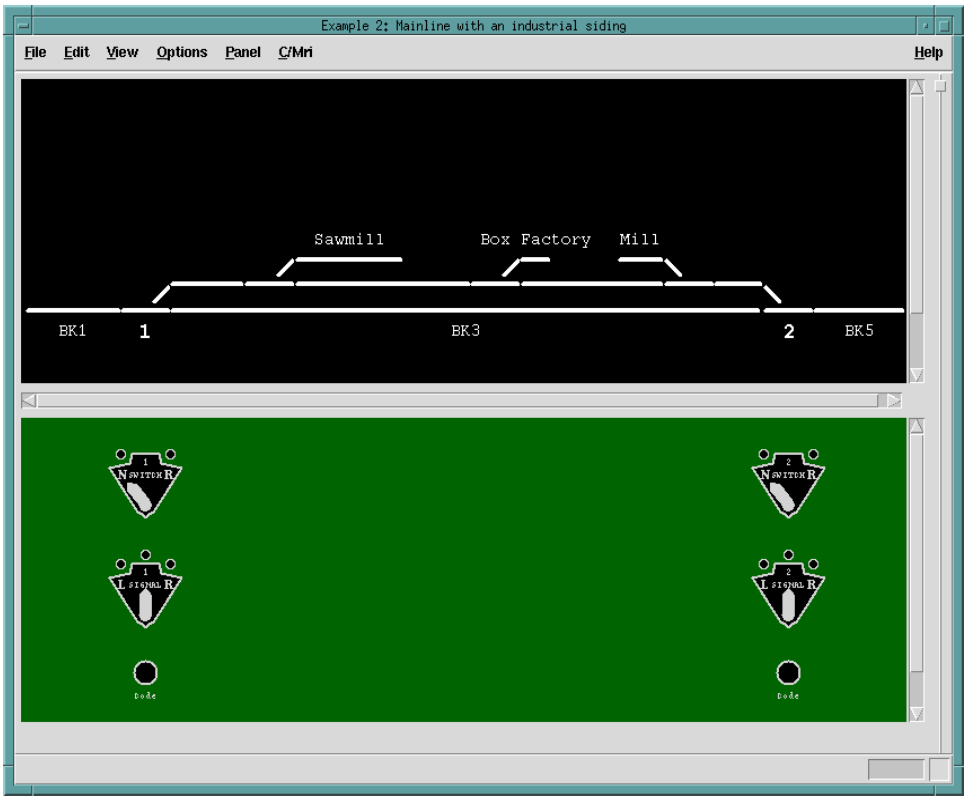


Figure 20.2 Example 2: Mainline with an industrial siding

Here is the code:

```
# Add User code after this line
# This is an example CTC Panel program.  It features a section of single
# tracked mainline with an industrial siding.
#
# The switches and signals are controled by a pair of SMINI nodes, which also
# connects occupation detectors and switch point sensors to the host system.
# The switches to the industrial sidings are manually operated (ground throws),
# with point position sensors.  There is no occupation detectors on the
# industrial sidings and their switches are not signaled.
# See example2.iow for the I/O connections for this example.
#
# The signals are two headed, three light signals.
#
package require snit;#          Make sure snit is loaded.
# Define types:
# This code uses SNIT to create a set of OO types to encapsulate each of
# the several elements of the system: trackwork sections (blocks),
# switches (turnouts), Switch Plates, Signal Plates, Signals, and control
# points.
#
namespace eval Blocks {
  # Block type (general trackwork).
  # Encapsulates block occupation detectors.
  #
  snit::type Block {
    # Occupation state values
    typevariable OCC 1
    typevariable CLR 0
    # Occupation state bit
    variable occupiedbit
    constructor {} {
      set occupiedbit $CLR;#    Initialize to clear.
    }
    # Occupation state methods
    method occupiedp {} {return [expr {$occupiedbit == $OCC}]}
    method setoccupied {value} {
```

```
      set occupiedbit $value
    }
  }
  # Main line trackage
  Block BK1;#   Block 1
  Block BK3;#   Block 3
  Block BK5;#   Block 5
  # Industrial siding sections
  Block ISide1
  Block ISide2
  Block ISide3
  Block ISide4
}
namespace eval Switches {
  # Switch type (turnout)
  # Encapsulates a switch (turnout), including its OS (delegated to a Block
  # object), its switch motor, and its point position sensor (its state).
  snit::type Switch {
    component block;#                   OS section
    delegate method * to block;#        Delegate block methods
    variable state unknown;#            Sense state (point position)
    # Motor bit values
    typevariable NOR 1;# 01
    typevariable REV 2;# 10
    variable motor;#                    Motor bits -- used to drive switch
#                                       motor.
    constructor {} {
      #                                 Install OS section
      install block using Blocks::Block %AUTO%
      # Initialize motor bits
      set motor $NOR
    }
    # State methods
    method getstate {} {return $state}
    method setstate {statebits} {
      if {$statebits == $NOR} {
        set state normal
      } elseif {$statebits == $REV} {
        set state reverse
      } else {
        set state unknown
      }
    }
    # Motor bit methods
    method motorbits {} {return $motor}
    method setmotor {mv} {
      switch -exact $mv {
        normal {set motor $NOR}
        reverse {set motor $REV}
      }
    }
  }
  Switch SW1;#          Switch at CP1
  Switch SW2;#          Switch at CP2
  Switch INDUS1;#       Switch at Sawmill
  Switch INDUS2;#       Switch at Box Factory
  Switch INDUS3;#       Switch at Mill
}
namespace eval SwitchPlates {
  # Switch Plate
  # Encapsulates a switch plate, implementing its lever position.
  snit::type SwitchPlate {
    component switch
    delegate method * to switch
    variable leverpos unknown
    constructor {sw} {
      set switch $sw
    }
    method setlever {pos} {set leverpos $pos}
    method getlever {} {return $leverpos}
  }
  SwitchPlate CP1SW Switches::SW1
  SwitchPlate CP2SW Switches::SW2
}
namespace eval Signals {
  # Signal types.  Encapsulates a signal's aspect.
  snit::type OneHead {
    # Single head signals have four states: dark, green, yellow, or red.
    typevariable aspects -array {
      Dark      0x00
      Green     0x01
```

```
      Yellow    0x02
      Red       0x04
    }
    variable aspectbits
    constructor {} {
      set aspectbits $aspects(Dark)
    }
    method setaspect {a} {set aspectbits $aspects($a)}
    method getaspect {}  {return $aspectbits}
  }
  snit::type TwoHead {
    # Two head signals have five states: dark, green over red,
    # yellow over red, red over yellow, and red over red.
    typevariable aspects -array {
      Dark      0x00
      GreenRed  0x11
      YellowRed 0x12
      RedYellow 0x0c
      RedRed    0x14
    }
    variable aspectbits
    constructor {} {
      set aspectbits $aspects(Dark)
    }
    method setaspect {a} {set aspectbits $aspects($a)}
    method getaspect {}  {return $aspectbits}
  }
  TwoHead CP1E;#        Heading into switch 1 from the west (block 1)
  TwoHead CP1WM;#       Heading into switch 1 from the east on the main (Block 3)
  TwoHead CP1WS;#       Heading into switch 1 from the east from the siding
  TwoHead CP2EM;#       Heading into switch 2 from the west on the main (Block 3)
  TwoHead CP2ES;#       Heading into switch 2 from the west from the siding
  TwoHead CP2W;#        Heading into switch 2 from the east (block 5)
}
namespace eval SignalPlates {
  # Signal Plate, encapsulating a signal plate with its lever and indicators.
  snit::type SignalPlate {
    variable leverpos unknown
    option -signalplate -default {} -readonly yes
    constructor {args} {
      $self configurelist $args
    }
    method setlever {pos} {set leverpos $pos}
    method getlever {} {return $leverpos}
    method setdot {dir} {
      switch $dir {
        left {
          MainWindow ctcpanel seti $options(-signalplate) L on
          MainWindow ctcpanel seti $options(-signalplate) C off
          MainWindow ctcpanel seti $options(-signalplate) R off
        }
        right {
          MainWindow ctcpanel seti $options(-signalplate) L off
          MainWindow ctcpanel seti $options(-signalplate) C off
          MainWindow ctcpanel seti $options(-signalplate) R on
        }
        none -
        default {
          MainWindow ctcpanel seti $options(-signalplate) L off
          MainWindow ctcpanel seti $options(-signalplate) C on
          MainWindow ctcpanel seti $options(-signalplate) R off
        }
      }
    }
  }
  SignalPlate CP1SG -signalplate CP1SG;#Signal plate for control point 1
  SignalPlate CP2SG -signalplate CP2SG;#Signal plate for control point 2
}
namespace eval ControlPoints {
  # Control points.  Used to implement code buttons.
  # Encapsulates a control point
  snit::type ControlPoint {
    option -cpname -readonly yes -default {}
    constructor {args} {
      $self configurelist $args
    }
    method code {} {
      foreach swp [MainWindow ctcpanel objectlist $options(-cpname) SwitchPlates] {
        MainWindow ctcpanel invoke $swp
      }
      foreach sgp [MainWindow ctcpanel objectlist $options(-cpname) SignalPlates] {
```

```
        MainWindow ctcpanel invoke $sgp
      }
    }
  }
  ControlPoint CP1 -cpname CP1;#        Control point 1
  ControlPoint CP2 -cpname CP2;#        Control point 2
}
# Main Loop Start
while {true} {
  # Read all ports
  set CP1_inbits [CP1 inputs]
  # Occupation Detectors and point state switches: (Input Port A, Card 1)
  set tempByte [lindex $CP1_inbits 0]
  Blocks::BK1  setoccupied  [expr {$tempByte      & 0x01}]
  Switches::SW1 setoccupied [expr {$tempByte » 1 & 0x01}]
  Switches::SW1 setstate    [expr {$tempByte » 2 & 0x03}]
  set CP2_inbits [CP2 inputs]
  # Occupation Detectors and point state switches: (Input Port A, Card 1)
  set tempByte [lindex $CP2_inbits 0]
  Blocks::BK5  setoccupied  [expr {$tempByte      & 0x01}]
  Switches::SW2 setoccupied [expr {$tempByte » 1 & 0x01}]
  Switches::SW2 setstate    [expr {$tempByte » 2 & 0x03}]
  set Siding_inbits [Siding inputs]
  # Occupation Detectors: (Input Port A, Card 1)
  set tempByte [lindex $Siding_inbits 0]
  Blocks::BK3    setoccupied    [expr {$tempByte      & 0x01}]
  Blocks::ISide1 setoccupied    [expr {$tempByte » 1 & 0x01}]
  Blocks::ISide2 setoccupied    [expr {$tempByte » 2 & 0x01}]
  Blocks::ISide3 setoccupied    [expr {$tempByte » 3 & 0x01}]
  Blocks::ISide4 setoccupied    [expr {$tempByte » 4 & 0x01}]
  Switches::INDUS1 setoccupied  [expr {$tempByte » 5 & 0x01}]
  Switches::INDUS2 setoccupied  [expr {$tempByte » 6 & 0x01}]
  Switches::INDUS3 setoccupied  [expr {$tempByte » 7 & 0x01}]
  # Point state switches: (Input Port B, Card 1)
  set tempByte [lindex $Siding_inbits 1]
  Switches::INDUS1 setstate     [expr {$tempByte      & 0x03}]
  Switches::INDUS2 setstate     [expr {$tempByte » 2 & 0x03}]
  Switches::INDUS3 setstate     [expr {$tempByte » 4 & 0x03}]
  # Invoke all trackwork and get occupicency
  MainWindow ctcpanel invoke BK1
  MainWindow ctcpanel invoke Bk3
  MainWindow ctcpanel invoke BK5
  MainWindow ctcpanel invoke ISide1
  MainWindow ctcpanel invoke ISide2
  MainWindow ctcpanel invoke ISide3
  MainWindow ctcpanel invoke ISide4
  MainWindow ctcpanel invoke INDUS1
  MainWindow ctcpanel invoke INDUS2
  MainWindow ctcpanel invoke INDUS3
  # Combine all siding ODs
  set Siding [expr {[Blocks::ISide1 occupiedp] ||
                    [Blocks::ISide2 occupiedp] ||
                    [Blocks::ISide3 occupiedp] ||
                    [Blocks::ISide4 occupiedp] ||
                    [Switches::INDUS1 occupiedp] ||
                    [Switches::INDUS2 occupiedp] ||
                    [Switches::INDUS3 occupiedp]}]
  # Initialize all signals to Red
  Signals::CP1E setaspect RedRed
  Signals::CP1WM setaspect RedRed
  Signals::CP1WS setaspect RedRed
  Signals::CP2EM setaspect RedRed
  Signals::CP2ES setaspect RedRed
  Signals::CP2W setaspect RedRed
  set dot1 none;# Assume no direction of travel through control point 1
  if {![MainWindow ctcpanel invoke SW1]} {
    # Switch1 OS is clear.  We can start to move the points
    Switches::SW1 setmotor [SwitchPlates::CP1SW getlever]
    # get current point state
    switch [Switches::SW1 getstate] {
      normal {;# Aligned for the Main.  Set the mainline signals.
        if {![Blocks::BK1 occupiedp] &&
            [string equal [SignalPlates::CP1SG getlever] "left"]} {
          # Clear to left
          Signals::CP1WM setaspect GreenRed
          set dot1 left
        }
        if {![Blocks::BK3 occupiedp] &&
            [string equal [SignalPlates::CP1SG getlever] "right"]} {
          # Clear to right
          Signals::CP1E setaspect GreenRed
```

```
        set dot1 right
      }
      # Set plate indicators
      MainWindow ctcpanel seti CP1SW N on
      MainWindow ctcpanel seti CP1SW R off
    }
    reverse {;# Aligned for the siding.  Set siding signals
      if {![Blocks::BK1 occupiedp] &&
          [string equal [SignalPlates::CP1SG getlever] "left"]} {
        # Clear to left
        Signals::CP1WS setaspect RedYellow
        set dot1 left
      }
      if {!$Siding && [string equal [SignalPlates::CP1SG getlever] "right"]} {
        # Clear to right
        Signals::CP1E setaspect RedYellow
        set dot1 right
      }
      # Set plate indicators
      MainWindow ctcpanel seti CP1SW N off
      MainWindow ctcpanel seti CP1SW R on
    }
    default {;# Points still moving.
      # Set plate indicators
      MainWindow ctcpanel seti CP1SW N off
      MainWindow ctcpanel seti CP1SW R off
    }
  }
}
# Set DOT on signal plate
SignalPlates::CP1SG setdot $dot1
# Switch 2 is much the same.
set dot2 none;# Assume no direction of travel through control point 1
if {![MainWindow ctcpanel invoke SW2]} {
  # Switch1 OS is clear.  We can start to move the points
  Switches::SW2 setmotor [SwitchPlates::CP2SW getlever]
  # get current point state
  switch [Switches::SW2 getstate] {
    normal {;# Aligned for the Main.  Set the mainline signals.
      if {![Blocks::BK5 occupiedp] &&
          [string equal [SignalPlates::CP2SG getlever] "right"]} {
        # Clear to right
        Signals::CP2EM setaspect GreenRed
        set dot2 right
      }
      if {![Blocks::BK3 occupiedp] &&
          [string equal [SignalPlates::CP2SG getlever] "left"]} {
        # Clear to left
        Signals::CP2W setaspect GreenRed
        set dot2 left
      }
      # Set plate indicators
      MainWindow ctcpanel seti CP2SW N on
      MainWindow ctcpanel seti CP2SW R off
    }
    reverse {;# Aligned for the siding.  Set siding signals
      if {![Blocks::BK5 occupiedp] &&
          [string equal [SignalPlates::CP2SG getlever] "right"]} {
        # Clear to right
        Signals::CP2ES setaspect RedYellow
        set dot2 right
      }
      if {!$Siding && [string equal [SignalPlates::CP2SG getlever] "left"]} {
        # Clear to left
        Signals::CP2W setaspect RedYellow
        set dot2 left
      }
      # Set plate indicators
      MainWindow ctcpanel seti CP2SW N off
      MainWindow ctcpanel seti CP2SW R on
    }
    default {;# Points still moving.
      # Set plate indicators
      MainWindow ctcpanel seti CP2SW N off
      MainWindow ctcpanel seti CP2SW R off
    }
  }
}
# Set DOT on signal plate
SignalPlates::CP2SG setdot $dot2
# Approach lighting -- darken signals facing empty blocks.
```

```
  if {![Blocks::BK1 occupiedp]} {
    Signals::CP1E setaspect Dark
  }
  if {![Blocks::BK3 occupiedp]} {
    Signals::CP1WM setaspect Dark
    Signals::CP2EM setaspect Dark
  }
  if {!$Siding} {
    Signals::CP1WS setaspect Dark
    Signals::CP2ES setaspect Dark
  }
  if {![Blocks::BK5 occupiedp]} {
    Signals::CP2W setaspect Dark
  }
  # Pack output bits
  # CP1:
  # Output Port A, Card 0 (CP1E and SW1)
  set CP1_outbits [expr {[Signals::CP1E   getaspect] | \
                         [Switches::SW1   motorbits] « 5}]
  # Output Port B, Card 0 (CP1WM)
  lappend CP1_outbits [Signals::CP1WM getaspect]
  # Output Port C, Card 0 (CP1WS)
  lappend CP1_outbits [Signals::CP1WS getaspect]
  # Output Ports A, B, C of Card 2: nothing
  lappend CP1_outbits 0 0 0
  # CP2:
  # Output Port A, Card 0 (CP2W and SW2)
  set CP2_outbits [expr {[Signals::CP2W   getaspect] | \
                         [Switches::SW2   motorbits] « 5}]
  # Output Port B, Card 0 (CP2EM)
  lappend CP2_outbits [Signals::CP2EM getaspect]
  # Output Port C, Card 0 (CP2ES)
  lappend CP2_outbits [Signals::CP2ES getaspect]
  # Output Ports A, B, C of Card 2: nothing
  lappend CP2_outbits 0 0 0
  # Write all output ports
  eval [list CP1 outputs] $CP1_outbits
  eval [list CP2 outputs] $CP2_outbits
  #eval [list Siding outputs] $Siding_outbits;# No Siding outputs ports used
  update;# Update display
}
# Main Loop End
```

And the I/O Worksheet:

```
SMINI @ UA 0 (CP1)
Card No. 0 Output
-------------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                    |
-------------------------------------------------------------------
|    | 0 | 1 | Upper Green    \                                   |
|    | 1 | 2 | Upper Yellow   | CP1E Signal                       |
|    | 2 | 3 | Upper Red      |                                   |
| A  | 3 | 4 | Lower Yellow   |                                   |
|    | 4 | 5 | Lower Red      /                                   |
|    | 5 | 6 | Normal   \  Switch 1                               |
|    | 6 | 7 | Reverse  /  motor                                  |
|    | 7 | 8 |                                                    |
-------------------------------------------------------------------
|    | 0 | 9 | Upper Green    \                                   |
|    | 1 |10 | Upper Yellow   | CP1WM Signal                      |
|    | 2 |11 | Upper Red      |                                   |
| B  | 3 |12 | Lower Yellow   |                                   |
|    | 4 |13 | Lower Red      /                                   |
|    | 5 |14 |                                                    |
|    | 6 |15 |                                                    |
|    | 7 |16 |                                                    |
-------------------------------------------------------------------
|    | 0 |17 | Upper Green    \                                   |
|    | 1 |18 | Upper Yellow   | CP1WS Signal                      |
|    | 2 |19 | Upper Red      |                                   |
| C  | 3 |20 | Lower Yellow   |                                   |
|    | 4 |21 | Lower Red      /                                   |
|    | 5 |22 |                                                    |
|    | 6 |23 |                                                    |
|    | 7 |24 |                                                    |
-------------------------------------------------------------------
|    | 0 |25 |                                                    |
|    | 1 |26 |                                                    |
|    | 2 |27 |                                                    |
| D  | 3 |28 |                                                    |
```

	4	29	
	5	30	
	6	31	
	7	32	

Card No. 1 Input

PORT	BIT	PIN	DESCRIPTION OF FUNCTION PERFORMED
	0	1	Block 1 Occupation Detector
	1	2	Switch 1 OS Occupation Detector
	2	3	Normal \ Switch 1
A	3	4	Reverse / state contacts
	4	5	
	5	6	
	6	7	
	7	8	
	0	9	
	1	10	
	2	11	
B	3	12	
	4	13	
	5	14	
	6	15	
	7	16	
	0	17	
	1	18	
	2	19	
C	3	20	
	4	21	
	5	22	
	6	23	
	7	24	
	0	25	
	1	26	
	2	27	
D	3	28	
	4	29	
	5	30	
	6	31	
	7	32	

Card No. 2 Output

PORT	BIT	PIN	DESCRIPTION OF FUNCTION PERFORMED
	0	1	
	1	2	
	2	3	
A	3	4	
	4	5	
	5	6	
	6	7	
	7	8	
	0	9	
	1	10	
	2	11	
B	3	12	
	4	13	
	5	14	
	6	15	
	7	16	
	0	17	
	1	18	
	2	19	
C	3	20	
	4	21	
	5	22	
	6	23	
	7	24	
	0	25	
	1	26	
	2	27	
D	3	28	
	4	29	

	5	30	
	6	31	
	7	32	

SMINI @ UA 1 (Siding)
Card No. 0 Output

PORT	BIT	PIN	DESCRIPTION OF FUNCTION PERFORMED
	0	1	
	1	2	
	2	3	
A	3	4	
	4	5	
	5	6	
	6	7	
	7	8	
	0	9	
	1	10	
	2	11	
B	3	12	
	4	13	
	5	14	
	6	15	
	7	16	
	0	17	
	1	18	
	2	19	
C	3	20	
	4	21	
	5	22	
	6	23	
	7	24	
	0	25	
	1	26	
	2	27	
D	3	28	
	4	29	
	5	30	
	6	31	
	7	32	

Card No. 1 Input

PORT	BIT	PIN	DESCRIPTION OF FUNCTION PERFORMED
	0	1	Block 3 Occupation Detector
	1	2	Siding, section 1 Occupation Detector
	2	3	Siding, section 2 Occupation Detector
A	3	4	Siding, section 3 Occupation Detector
	4	5	Siding, section 4 Occupation Detector
	5	6	Industry 1 Occupation Detector
	6	7	Industry 2 Occupation Detector
	7	8	Industry 3 Occupation Detector
	0	9	Normal \ Manual switch
	1	10	Reverse / Industry 1 state
	2	11	Normal \ Manual switch
B	3	12	Reverse / Industry 2 state
	4	13	Normal \ Manual switch
	5	14	Reverse / Industry 3 state
	6	15	
	7	16	
	0	17	
	1	18	
	2	19	
C	3	20	
	4	21	
	5	22	
	6	23	
	7	24	
	0	25	
	1	26	
	2	27	
D	3	28	
	4	29	

```
|    | 5 |30 |                                                    |
|    | 6 |31 |                                                    |
|    | 7 |32 |                                                    |
-------------------------------------------------------------------
Card No. 2 Output
-------------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                   |
-------------------------------------------------------------------
|    | 0 | 1 |                                                    |
|    | 1 | 2 |                                                    |
|    | 2 | 3 |                                                    |
| A  | 3 | 4 |                                                    |
|    | 4 | 5 |                                                    |
|    | 5 | 6 |                                                    |
|    | 6 | 7 |                                                    |
|    | 7 | 8 |                                                    |
-------------------------------------------------------------------
|    | 0 | 9 |                                                    |
|    | 1 |10 |                                                    |
|    | 2 |11 |                                                    |
| B  | 3 |12 |                                                    |
|    | 4 |13 |                                                    |
|    | 5 |14 |                                                    |
|    | 6 |15 |                                                    |
|    | 7 |16 |                                                    |
-------------------------------------------------------------------
|    | 0 |17 |                                                    |
|    | 1 |18 |                                                    |
|    | 2 |19 |                                                    |
| C  | 3 |20 |                                                    |
|    | 4 |21 |                                                    |
|    | 5 |22 |                                                    |
|    | 6 |23 |                                                    |
|    | 7 |24 |                                                    |
-------------------------------------------------------------------
|    | 0 |25 |                                                    |
|    | 1 |26 |                                                    |
|    | 2 |27 |                                                    |
| D  | 3 |28 |                                                    |
|    | 4 |29 |                                                    |
|    | 5 |30 |                                                    |
|    | 6 |31 |                                                    |
|    | 7 |32 |                                                    |
-------------------------------------------------------------------
SMINI @ UA 2 (CP2)
Card No. 0 Output
-------------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                   |
-------------------------------------------------------------------
|    | 0 | 1 | Upper Green    \                                   |
|    | 1 | 2 | Upper Yellow   | CP2W Signal                       |
|    | 2 | 3 | Upper Red      |                                   |
| A  | 3 | 4 | Lower Yellow   |                                   |
|    | 4 | 5 | Lower Red      /                                   |
|    | 5 | 6 | Normal   \  Switch 2                               |
|    | 6 | 7 | Reverse  /  motor                                  |
|    | 7 | 8 |                                                    |
-------------------------------------------------------------------
|    | 0 | 9 | Upper Green    \                                   |
|    | 1 |10 | Upper Yellow   | CP2EM Signal                      |
|    | 2 |11 | Upper Red      |                                   |
| B  | 3 |12 | Lower Yellow   |                                   |
|    | 4 |13 | Lower Red      /                                   |
|    | 5 |14 |                                                    |
|    | 6 |15 |                                                    |
|    | 7 |16 |                                                    |
-------------------------------------------------------------------
|    | 0 |17 | Upper Green    \                                   |
|    | 1 |18 | Upper Yellow   | CP2ES Signal                      |
|    | 2 |19 | Upper Red      |                                   |
| C  | 3 |20 | Lower Yellow   |                                   |
|    | 4 |21 | Lower Red      /                                   |
|    | 5 |22 |                                                    |
|    | 6 |23 |                                                    |
|    | 7 |24 |                                                    |
-------------------------------------------------------------------
|    | 0 |25 |                                                    |
|    | 1 |26 |                                                    |
|    | 2 |27 |                                                    |
| D  | 3 |28 |                                                    |
|    | 4 |29 |                                                    |
```

	5	30		
	6	31		
	7	32		

Card No. 1 Input

PORT	BIT	PIN	DESCRIPTION OF FUNCTION PERFORMED
	0	1	Block 5 Occupation Detector
	1	2	Switch 2 OS Occupation Detector
	2	3	Normal \ Switch 2
A	3	4	Reverse / state contacts
	4	5	
	5	6	
	6	7	
	7	8	
	0	9	
	1	10	
	2	11	
B	3	12	
	4	13	
	5	14	
	6	15	
	7	16	
	0	17	
	1	18	
	2	19	
C	3	20	
	4	21	
	5	22	
	6	23	
	7	24	
	0	25	
	1	26	
	2	27	
D	3	28	
	4	29	
	5	30	
	6	31	
	7	32	

Card No. 2 Output

PORT	BIT	PIN	DESCRIPTION OF FUNCTION PERFORMED
	0	1	
	1	2	
	2	3	
A	3	4	
	4	5	
	5	6	
	6	7	
	7	8	
	0	9	
	1	10	
	2	11	
B	3	12	
	4	13	
	5	14	
	6	15	
	7	16	
	0	17	
	1	18	
	2	19	
C	3	20	
	4	21	
	5	22	
	6	23	
	7	24	
	0	25	
	1	26	
	2	27	
D	3	28	
	4	29	
	5	30	

```
|     | 6 |31 |                                                    |
|     | 7 |32 |                                                    |
-------------------------------------------------------------
EOF
```

20.3 Example 3: double track crossover

This example, shown below, implements a double track crossover. Uses two SMINI boards, one for each of the two control points.

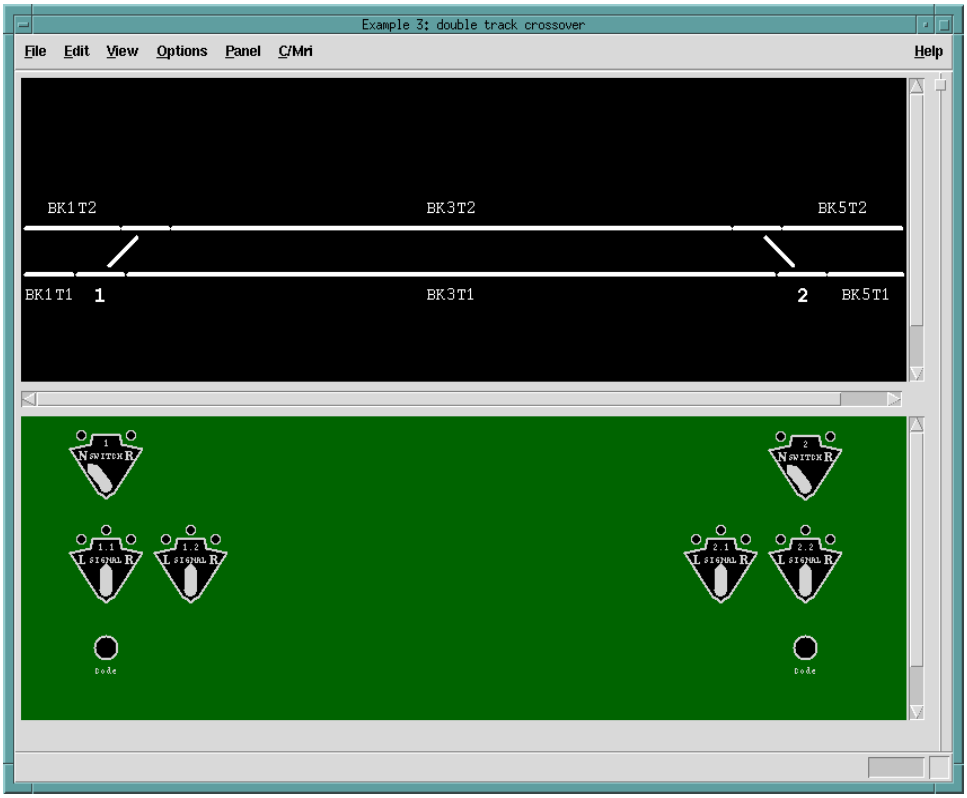


Figure 20.3 Example 3: Double track crossover

Here is the code:

```
# Add User code after this line
# This is an example CTC Panel program.  It features a section of double
# tracked mainline with a pair of crossovers
#
# The switches and signals are controled by a pair of SMINI nodes, which also
# connects occupation detectors and switch point sensors to the host system.
# See example3.iow for I/O connections for this example.
#
# The signals are two headed, three light signals.
#
package require snit;#          Make sure snit is loaded.
# Define types:
# This code uses SNIT to create a set of OO types to encapsulate each of
# the several elements of the system: trackwork sections (blocks),
# switches (turnouts), Switch Plates, Signal Plates, Signals, and control
# points.
#
namespace eval Blocks {
```

```
  # Block type (general trackwork).
  # Encapsulates block occupation detectors.
  #
  snit::type Block {
    # Occupation state values
    typevariable OCC 1
    typevariable CLR 0
    # Occupation state bit
    variable occupiedbit
    constructor {} {
      set occupiedbit $CLR;#    Initialize to clear.
    }
    # Occupation state methods
    method occupiedp {} {return [expr {$occupiedbit == $OCC}]}
    method setoccupied {value} {
      set occupiedbit $value
    }
  }
  # Main line trackage
  Block BK1T1;# Block 1, East
  Block BK1T2;# Block 1, West
  Block BK3T1;# Block 3, East
  Block BK3T2;# Block 3, West
  Block BK5T1;# Block 5, East
  Block BK5T2;# Block 5, West
}
namespace eval Switches {
  # Switch type (turnout)
  # Encapsulates a switch (turnout), including its OS (delegated to a Block
  # object), its switch motor, and its point position sensor (its state).
  snit::type Switch {
    component block;#                   OS section
    delegate method * to block;#        Delegate block methods
    variable state unknown;#            Sense state (point position)
    # Motor bit values
    typevariable NOR 1;# 01
    typevariable REV 2;# 10
    variable motor;#                    Motor bits -- used to drive switch
#                                       motor.
    constructor {} {
      #                                 Install OS section
      install block using Blocks::Block %AUTO%
      # Initialize motor bits
      set motor $NOR
    }
    # State methods
    method getstate {} {return $state}
    method setstate {statebits} {
      if {$statebits == $NOR} {
        set state normal
      } elseif {$statebits == $REV} {
        set state reverse
      } else {
        set state unknown
      }
    }
    # Motor bit methods
    method motorbits {} {return $motor}
    method setmotor {mv} {
      switch -exact $mv {
        normal {set motor $NOR}
        reverse {set motor $REV}
      }
    }
  }
  Switch SW1a;#         Switch at CP1
  Switch SW1b;#         Switch at CP1
  Switch SW2a;#         Switch at CP2
  Switch SW2b;#         Switch at CP2
}
namespace eval SwitchPlates {
  # Switch Plate
  # Encapsulates a switch plate, implementing its lever position.
  snit::type SwitchPlate {
    delegate method * to switch
    variable leverpos unknown
    constructor {sw} {
      set switch $sw
    }
    method setlever {pos} {set leverpos $pos}
    method getlever {} {return $leverpos}
```

```
  }
  SwitchPlate CP1SW
  SwitchPlate CP2SW
}
namespace eval Signals {
  # Signal types.  Encapsulates a signal's aspect.
  snit::type OneHead {
    # Single head signals have four states: dark, green, yellow, or red.
    typevariable aspects -array {
      Dark      0x00
      Green     0x01
      Yellow    0x02
      Red       0x04
    }
    variable aspectbits
    constructor {} {
      set aspectbits $aspects(Dark)
    }
    method setaspect {a} {set aspectbits $aspects($a)}
    method getaspect {}  {return $aspectbits}
  }
  snit::type TwoHead {
    # Two head signals have five states: dark, green over red,
    # yellow over red, red over yellow, and red over red.
    typevariable aspects -array {
      Dark      0x00
      GreenRed  0x11
      YellowRed 0x12
      RedYellow 0x0c
      RedRed    0x14
    }
    variable aspectbits
    constructor {} {
      set aspectbits $aspects(Dark)
    }
    method setaspect {a} {set aspectbits $aspects($a)}
    method getaspect {}  {return $aspectbits}
  }
  TwoHead CP1ET1;#      Heading into switch 1 from the west (block 1, Track 1)
  TwoHead CP1WT1;#      Heading into switch 1 from the east (Block 3, Track 1)
  TwoHead CP1ET2;#      Heading into switch 1 from the east (block 1, Track 2)
  TwoHead CP1WT2;#      Heading into switch 1 from the west (Block 3, Track 2)
  TwoHead CP2ET1;#      Heading into switch 1 from the west (block 3, Track 1)
  TwoHead CP2WT1;#      Heading into switch 1 from the east (Block 5, Track 1)
  TwoHead CP2ET2;#      Heading into switch 1 from the east (block 3, Track 2)
  TwoHead CP2WT2;#      Heading into switch 1 from the west (Block 5, Track 2)
}
namespace eval SignalPlates {
  # Signal Plate, encapsulating a signal plate with its lever and indicators.
  snit::type SignalPlate {
    variable leverpos unknown
    option -signalplate -default {} -readonly yes
    constructor {args} {
      $self configurelist $args
    }
    method setlever {pos} {set leverpos $pos}
    method getlever {} {return $leverpos}
    method setdot {dir} {
      switch $dir {
        left {
          MainWindow ctcpanel seti $options(-signalplate) L on
          MainWindow ctcpanel seti $options(-signalplate) C off
          MainWindow ctcpanel seti $options(-signalplate) R off
        }
        right {
          MainWindow ctcpanel seti $options(-signalplate) L off
          MainWindow ctcpanel seti $options(-signalplate) C off
          MainWindow ctcpanel seti $options(-signalplate) R on
        }
        none -
        default {
          MainWindow ctcpanel seti $options(-signalplate) L off
          MainWindow ctcpanel seti $options(-signalplate) C on
          MainWindow ctcpanel seti $options(-signalplate) R off
        }
      }
    }
  }
  SignalPlate CP1T1SG -signalplate CP1T1SG;#Signal plate for CP1, track 1
  SignalPlate CP1T2SG -signalplate TRACK2;#Signal plate for CP1, track 2
  SignalPlate CP2T1SG -signalplate CP1T1SG;#Signal plate for CP2, track 1
```

```
  SignalPlate CP2T2SG -signalplate TRACK2;#Signal plate for CP2, track 2
}
namespace eval ControlPoints {
  # Control points.  Used to implement code buttons.
  # Encapsulates a control point
  snit::type ControlPoint {
    option -cpname -readonly yes -default {}
    constructor {args} {
      $self configurelist $args
    }
    method code {} {
      foreach swp [MainWindow ctcpanel objectlist $options(-cpname) SwitchPlates] {
        MainWindow ctcpanel invoke $swp
      }
      foreach sgp [MainWindow ctcpanel objectlist $options(-cpname) SignalPlates] {
        MainWindow ctcpanel invoke $sgp
      }
    }
  }
  ControlPoint CP1 -cpname CP1;#        Control point 1
  ControlPoint CP2 -cpname CP2;#        Control point 2
}
# Main Loop Start
while {true} {
  # Read all ports
  set CP1_inbits [CP1 inputs]
  # Occupation Detectors and point state switches: (Input Port A)
  set tempByte [lindex $CP1_inbits 0]
  Blocks::BK1T1 setoccupied  [expr {$tempByte      & 0x01}]
  Blocks::BK1T2 setoccupied  [expr {$tempByte » 1 & 0x01}]
  Blocks::BK3T1 setoccupied  [expr {$tempByte » 2 & 0x01}]
  # The both switches are on a common OD for their OSs
  Switches::SW1a setoccupied [expr {$tempByte » 3 & 0x01}]
  Switches::SW1b setoccupied [expr {$tempByte » 3 & 0x01}]
  Switches::SW1a setstate    [expr {$tempByte » 4 & 0x03}]
  Switches::SW1b setstate    [expr {$tempByte » 6 & 0x03}]
  set CP2_inbits [CP2 inputs]
  # Occupation Detectors and point state switches: (Input Port A)
  set tempByte [lindex $CP2_inbits 0]
  Blocks::BK5T1 setoccupied  [expr {$tempByte      & 0x01}]
  Blocks::BK5T2 setoccupied  [expr {$tempByte » 1 & 0x01}]
  Blocks::BK3T2 setoccupied  [expr {$tempByte » 2 & 0x01}]
  # The both switches are on a common OD for their OSs
  Switches::SW2a setoccupied [expr {$tempByte » 3 & 0x01}]
  Switches::SW2b setoccupied [expr {$tempByte » 3 & 0x01}]
  Switches::SW2a setstate    [expr {$tempByte » 4 & 0x03}]
  Switches::SW2b setstate    [expr {$tempByte » 6 & 0x03}]
  # Invoke all trackwork and get occupicency
  MainWindow ctcpanel invoke BK1T1
  MainWindow ctcpanel invoke BK1T2
  MainWindow ctcpanel invoke BK3T1
  MainWindow ctcpanel invoke BK3T2
  MainWindow ctcpanel invoke BK5T1
  MainWindow ctcpanel invoke BK5T2
  MainWindow ctcpanel invoke SW1a
  MainWindow ctcpanel invoke SW1b
  MainWindow ctcpanel invoke SW2a
  MainWindow ctcpanel invoke SW2b
  # Initialize all signals to Red
  Signals::CP1ET1 setaspect RedRed
  Signals::CP1WT1 setaspect RedRed
  Signals::CP1ET2 setaspect RedRed
  Signals::CP1WT2 setaspect RedRed
  Signals::CP2ET1 setaspect RedRed
  Signals::CP2WT1 setaspect RedRed
  Signals::CP2ET2 setaspect RedRed
  Signals::CP2WT2 setaspect RedRed
  set dot1t1 none ;# Assume no direction of travel through control point 1, track 1
  set dot1t2 none ;# Assume no direction of travel through control point 1, track 2
  if {![MainWindow ctcpanel invoke SW1a] &&
      ![MainWindow ctcpanel invoke SW1b]} {
    # Switch1 (a & b) OS is clear.  We can start to move the points
    Switches::SW1a setmotor [SwitchPlates::CP1SW getlever]
    Switches::SW1b setmotor [SwitchPlates::CP1SW getlever]
    set s1a [Switches::SW1a getstate]
    set s1b [Switches::SW1b getstate]
    if {[string equal "$s1a" "$s1b"]} {
      switch $s1a {
        normal {;# Aligned for the Main.  Set the mainline signals.
          if {![Blocks::BK1T1 occupiedp] &&
              [string equal [SignalPlates::CP1T1SG getlever] "left"]} {
```

```
          # Clear to left on track 1
          Signals::CP1WT1 setaspect GreenRed
          set dot1t1 left
        }
        if {![Blocks::BK1T2 occupiedp] &&
            [string equal [SignalPlates::CP1T2SG getlever] "left"]} {
          # Clear to left on track 2
          Signals::CP1WT2 setaspect GreenRed
          set dot1t2 left
        }
        if {![Blocks::BK3T1 occupiedp] &&
            [string equal [SignalPlates::CP1T1SG getlever] "right"]} {
          # Clear to right on track 1
          Signals::CP1ET1 setaspect GreenRed
          set dot1t1 right
        }
        if {![Blocks::BK3T2 occupiedp] &&
            [string equal [SignalPlates::CP1T2SG getlever] "right"]} {
          # Clear to right on track 2
          Signals::CP1ET2 setaspect GreenRed
          set dot1t2 right
        }
        # Set plate indicators
        MainWindow ctcpanel seti CP1SW N on
        MainWindow ctcpanel seti CP1SW R off
      }
      reverse {;# Aligned for crossing over
        if {[string equal [SignalPlates::CP1T1SG getlever] [SignalPlates::CP1T2SG getlever]]} {
          if {![Blocks::BK1T1 occupiedp] &&
               [string equal [SignalPlates::CP1T1SG getlever] "left"]} {
            Signals::CP1WT2 setaspect RedYellow
            set dot1t1 left
            set dot1t2 left
          }
          if {![Blocks::BK3T2 occupiedp] &&
              [string equal [SignalPlates::CP1T2SG getlever] "right"]} {
            Signals::CP1ET1 setaspect RedYellow
            set dot1t1 right
            set dot1t2 right
          }
        }
        # Set plate indicators
        MainWindow ctcpanel seti CP1SW N off
        MainWindow ctcpanel seti CP1SW R on
      }
      default {;# Points still moving.
        # Set plate indicators
        MainWindow ctcpanel seti CP1SW N off
        MainWindow ctcpanel seti CP1SW R off
      }
    }
  } else {;# Points not consistently aligned
    # Set plate indicators
    MainWindow ctcpanel seti CP1SW N off
    MainWindow ctcpanel seti CP1SW R off
  }
}
# Set DOT on signal plates
SignalPlates::CP1T1SG setdot $dot1t1
SignalPlates::CP1T2SG setdot $dot1t2
set dot2t1 none ;# Assume no direction of travel through control point 2, track 1
set dot2t2 none ;# Assume no direction of travel through control point 2, track 2
if {![MainWindow ctcpanel invoke SW2a] &&
    ![MainWindow ctcpanel invoke SW2b]} {
  # Switch2 (a & b) OS is clear.  We can start to move the points
  Switches::SW2a setmotor [SwitchPlates::CP2SW getlever]
  Switches::SW2b setmotor [SwitchPlates::CP2SW getlever]
  set s2a [Switches::SW2a getstate]
  set s2b [Switches::SW2b getstate]
  if {[string equal "$s2a" "$s2b"]} {
    switch $s2a {
      normal {;# Aligned for the Main.  Set the mainline signals.
        if {![Blocks::BK3T1 occupiedp] &&
            [string equal [SignalPlates::CP2T1SG getlever] "left"]} {
          # Clear to left on track 1
          Signals::CP2WT1 setaspect GreenRed
          set dot2t1 left
        }
        if {![Blocks::BK3T2 occupiedp] &&
            [string equal [SignalPlates::CP2T2SG getlever] "left"]} {
          # Clear to left on track 2
```

```
            Signals::CP2WT2 setaspect GreenRed
            set dot2t2 left
          }
          if {![Blocks::BK5T1 occupiedp] &&
              [string equal [SignalPlates::CP2T1SG getlever] "right"]} {
            # Clear to right on track 1
            Signals::CP2ET1 setaspect GreenRed
            set dot2t1 right
          }
          if {![Blocks::BK5T2 occupiedp] &&
              [string equal [SignalPlates::CP2T2SG getlever] "right"]} {
            # Clear to right on track 2
            Signals::CP2ET2 setaspect GreenRed
            set dot2t2 right
          }
          # Set plate indicators
          MainWindow ctcpanel seti CP2SW N on
          MainWindow ctcpanel seti CP2SW R off
        }
        reverse {;# Aligned for crossing over
          if {[string equal [SignalPlates::CP2T1SG getlever] [SignalPlates::CP2T2SG getlever]]} {
            if {![Blocks::BK3T2 occupiedp] &&
                 [string equal [SignalPlates::CP2T2SG getlever] "left"]} {
              Signals::CP2WT1 setaspect RedYellow
              set dot2t1 left
              set dot2t2 left
            }
            if {![Blocks::BK5T1 occupiedp] &&
                [string equal [SignalPlates::CP2T1SG getlever] "right"]} {
              Signals::CP2ET2 setaspect RedYellow
              set dot2t1 right
              set dot2t2 right
            }
          }
          # Set plate indicators
          MainWindow ctcpanel seti CP2SW N off
          MainWindow ctcpanel seti CP2SW R on
        }
        default {;# Points still moving.
          # Set plate indicators
          MainWindow ctcpanel seti CP2SW N off
          MainWindow ctcpanel seti CP2SW R off
        }
      }
    } else {;# Points not consistently aligned
      # Set plate indicators
      MainWindow ctcpanel seti CP2SW N off
      MainWindow ctcpanel seti CP2SW R off
    }
  }
  # Set DOT on signal plates
  SignalPlates::CP2T1SG setdot $dot2t1
  SignalPlates::CP2T2SG setdot $dot2t2
  # Approach lighting -- darken signals facing empty blocks.
  if {![Blocks::BK1T1 occupiedp]} {
    Signals::CP1ET1 setaspect Dark
  }
  if {![Blocks::BK1T2 occupiedp]} {
    Signals::CP1ET2 setaspect Dark
  }
  if {![Blocks::BK3T1 occupiedp]} {
    Signals::CP1WT1 setaspect Dark
    Signals::CP2ET1 setaspect Dark
  }
  if {![Blocks::BK1T2 occupiedp]} {
    Signals::CP1WT2 setaspect Dark
    Signals::CP2ET2 setaspect Dark
  }
  if {![Blocks::BK5T1 occupiedp]} {
    Signals::CP2WT1 setaspect Dark
  }
  if {![Blocks::BK5T2 occupiedp]} {
    Signals::CP2WT2 setaspect Dark
  }
  # Pack output bits
  # CP1:
  set     CP1_outbits [Signals::CP1ET1 getaspect];# Output Port A, Card 0 (CP1ET1)
  lappend CP1_outbits [Signals::CP1ET2 getaspect];# Output Port B, Card 0 (CP1ET2)
  lappend CP1_outbits [Signals::CP1WT1 getaspect];# Output Port C, Card 0 (CP1WT1)
  lappend CP1_outbits [Signals::CP1WT2 getaspect];# Output Port A, Card 2 (CP1WT2)
  # Output Port B, Card 2 (Switch motor bits SW1a and SW1b)
```

```
  lappend CP1_outbits [expr {[Switches::SW1a motorbits] | \
                             [Switches::SW1b motorbits] « 2}]
  lappend CP1_outbits 0;#                          Output Port C, Card 2 (unused)
  # CP2:
  set     CP2_outbits [Signals::CP2ET1 getaspect];# Output Port A, Card 0 (CP2ET1)
  lappend CP2_outbits [Signals::CP2ET2 getaspect];# Output Port B, Card 0 (CP2ET2)
  lappend CP2_outbits [Signals::CP2WT1 getaspect];# Output Port C, Card 0 (CP2WT1)
  lappend CP2_outbits [Signals::CP2WT2 getaspect];# Output Port A, Card 2 (CP2WT2)
  # Output Port B, Card 2 (Switch motor bits SW2a and SW2b)
  lappend CP2_outbits [expr {[Switches::SW2a motorbits] | \
                             [Switches::SW2b motorbits] « 2}]
  lappend CP2_outbits 0;#                          Output Port C, Card 2 (unused)
  # Write all output ports
  eval [list CP1 outputs] $CP1_outbits
  eval [list CP2 outputs] $CP2_outbits
  update;# Update display
}
# Main Loop End
```

And the I/O Worksheet:

```
SMINI @ UA 0 (CP1)
Card No. 0 Output
-----------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                |
-----------------------------------------------------------------
|    | 0 | 1 | Upper Green    \                               |
|    | 1 | 2 | Upper Yellow   | CP1ET1 Signal                 |
|    | 2 | 3 | Upper Red      |                               |
| A  | 3 | 4 | Lower Yellow   |                               |
|    | 4 | 5 | Lower Red      /                               |
|    | 5 | 6 |                                                |
|    | 6 | 7 |                                                |
|    | 7 | 8 |                                                |
-----------------------------------------------------------------
|    | 0 | 9 | Upper Green    \                               |
|    | 1 |10 | Upper Yellow   | CP1ET2 Signal                 |
|    | 2 |11 | Upper Red      |                               |
| B  | 3 |12 | Lower Yellow   |                               |
|    | 4 |13 | Lower Red      /                               |
|    | 5 |14 |                                                |
|    | 6 |15 |                                                |
|    | 7 |16 |                                                |
-----------------------------------------------------------------
|    | 0 |17 | Upper Green    \                               |
|    | 1 |18 | Upper Yellow   | CP1WT1 Signal                 |
|    | 2 |19 | Upper Red      |                               |
| C  | 3 |20 | Lower Yellow   |                               |
|    | 4 |21 | Lower Red      /                               |
|    | 5 |22 |                                                |
|    | 6 |23 |                                                |
|    | 7 |24 |                                                |
-----------------------------------------------------------------
|    | 0 |25 |                                                |
|    | 1 |26 |                                                |
|    | 2 |27 |                                                |
| D  | 3 |28 |                                                |
|    | 4 |29 |                                                |
|    | 5 |30 |                                                |
|    | 6 |31 |                                                |
|    | 7 |32 |                                                |
-----------------------------------------------------------------
Card No. 1 Input
-----------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                |
-----------------------------------------------------------------
|    | 0 | 1 | Block 1, Track 1 Occupation Detector           |
|    | 1 | 2 | Block 1, Track 2 Occupation Detector           |
|    | 2 | 3 | Block 3, Track 1 Occupation Detector           |
| A  | 3 | 4 | OS 1 Occupation Detector                       |
|    | 4 | 5 | Normal  \  Switch 1a                           |
|    | 5 | 6 | Reverse /  state contacts                      |
|    | 6 | 7 | Normal  \  Switch 1a                           |
|    | 7 | 8 | Reverse /  state contacts                      |
-----------------------------------------------------------------
|    | 0 | 9 |                                                |
|    | 1 |10 |                                                |
|    | 2 |11 |                                                |
| B  | 3 |12 |                                                |
|    | 4 |13 |                                                |
|    | 5 |14 |                                                |
```

```
|     | 6 |15 |                                                    |
|     | 7 |16 |                                                    |
-------------------------------------------------------------------
|     | 0 |17 |                                                    |
|     | 1 |18 |                                                    |
|     | 2 |19 |                                                    |
| C   | 3 |20 |                                                    |
|     | 4 |21 |                                                    |
|     | 5 |22 |                                                    |
|     | 6 |23 |                                                    |
|     | 7 |24 |                                                    |
-------------------------------------------------------------------
|     | 0 |25 |                                                    |
|     | 1 |26 |                                                    |
|     | 2 |27 |                                                    |
| D   | 3 |28 |                                                    |
|     | 4 |29 |                                                    |
|     | 5 |30 |                                                    |
|     | 6 |31 |                                                    |
|     | 7 |32 |                                                    |
-------------------------------------------------------------------
Card No. 2 Output
-------------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                   |
-------------------------------------------------------------------
|     | 0 | 1 | Upper Green    \                                   |
|     | 1 | 2 | Upper Yellow   | CP1WT2 Signal                     |
|     | 2 | 3 | Upper Red      |                                   |
| A   | 3 | 4 | Lower Yellow   |                                   |
|     | 4 | 5 | Lower Red      /                                   |
|     | 5 | 6 |                                                    |
|     | 6 | 7 |                                                    |
|     | 7 | 8 |                                                    |
-------------------------------------------------------------------
|     | 0 | 9 | Normal   \  Switch 1a                              |
|     | 1 |10 | Reverse  /  motor                                  |
|     | 2 |11 | Normal   \  Switch 1b                              |
| B   | 3 |12 | Reverse  /  motor                                  |
|     | 4 |13 |                                                    |
|     | 5 |14 |                                                    |
|     | 6 |15 |                                                    |
|     | 7 |16 |                                                    |
-------------------------------------------------------------------
|     | 0 |17 |                                                    |
|     | 1 |18 |                                                    |
|     | 2 |19 |                                                    |
| C   | 3 |20 |                                                    |
|     | 4 |21 |                                                    |
|     | 5 |22 |                                                    |
|     | 6 |23 |                                                    |
|     | 7 |24 |                                                    |
-------------------------------------------------------------------
|     | 0 |25 |                                                    |
|     | 1 |26 |                                                    |
|     | 2 |27 |                                                    |
| D   | 3 |28 |                                                    |
|     | 4 |29 |                                                    |
|     | 5 |30 |                                                    |
|     | 6 |31 |                                                    |
|     | 7 |32 |                                                    |
-------------------------------------------------------------------
SMINI @ UA 1 (CP2)
Card No. 0 Output
-------------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                   |
-------------------------------------------------------------------
|     | 0 | 1 | Upper Green    \                                   |
|     | 1 | 2 | Upper Yellow   | CP2ET1 Signal                     |
|     | 2 | 3 | Upper Red      |                                   |
| A   | 3 | 4 | Lower Yellow   |                                   |
|     | 4 | 5 | Lower Red      /                                   |
|     | 5 | 6 |                                                    |
|     | 6 | 7 |                                                    |
|     | 7 | 8 |                                                    |
-------------------------------------------------------------------
|     | 0 | 9 | Upper Green    \                                   |
|     | 1 |10 | Upper Yellow   | CP2ET2 Signal                     |
|     | 2 |11 | Upper Red      |                                   |
| B   | 3 |12 | Lower Yellow   |                                   |
|     | 4 |13 | Lower Red      /                                   |
|     | 5 |14 |                                                    |
```

```
|    | 6 |15 |                                                    |
|    | 7 |16 |                                                    |
-------------------------------------------------------------------
|    | 0 |17 | Upper Green    \                                   |
|    | 1 |18 | Upper Yellow   | CP2WT1 Signal                     |
|    | 2 |19 | Upper Red      |                                   |
| C  | 3 |20 | Lower Yellow   |                                   |
|    | 4 |21 | Lower Red      /                                   |
|    | 5 |22 |                                                    |
|    | 6 |23 |                                                    |
|    | 7 |24 |                                                    |
-------------------------------------------------------------------
|    | 0 |25 |                                                    |
|    | 1 |26 |                                                    |
|    | 2 |27 |                                                    |
| D  | 3 |28 |                                                    |
|    | 4 |29 |                                                    |
|    | 5 |30 |                                                    |
|    | 6 |31 |                                                    |
|    | 7 |32 |                                                    |
-------------------------------------------------------------------
Card No. 1 Input
-------------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                   |
-------------------------------------------------------------------
|    | 0 | 1 | Block 5, Track 1 Occupation Detector               |
|    | 1 | 2 | Block 5, Track 2 Occupation Detector               |
|    | 2 | 3 | Block 3, Track 2 Occupation Detector               |
| A  | 3 | 4 | OS 2 Occupation Detector                           |
|    | 4 | 5 | Normal  \  Switch 2a                               |
|    | 5 | 6 | Reverse /  state contacts                          |
|    | 6 | 7 | Normal  \  Switch 2b                               |
|    | 7 | 8 | Reverse /  state contacts                          |
-------------------------------------------------------------------
|    | 0 | 9 |                                                    |
|    | 1 |10 |                                                    |
|    | 2 |11 |                                                    |
| B  | 3 |12 |                                                    |
|    | 4 |13 |                                                    |
|    | 5 |14 |                                                    |
|    | 6 |15 |                                                    |
|    | 7 |16 |                                                    |
-------------------------------------------------------------------
|    | 0 |17 |                                                    |
|    | 1 |18 |                                                    |
|    | 2 |19 |                                                    |
| C  | 3 |20 |                                                    |
|    | 4 |21 |                                                    |
|    | 5 |22 |                                                    |
|    | 6 |23 |                                                    |
|    | 7 |24 |                                                    |
-------------------------------------------------------------------
|    | 0 |25 |                                                    |
|    | 1 |26 |                                                    |
|    | 2 |27 |                                                    |
| D  | 3 |28 |                                                    |
|    | 4 |29 |                                                    |
|    | 5 |30 |                                                    |
|    | 6 |31 |                                                    |
|    | 7 |32 |                                                    |
-------------------------------------------------------------------
Card No. 2 Output
-------------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                   |
-------------------------------------------------------------------
|    | 0 | 1 | Upper Green    \                                   |
|    | 1 | 2 | Upper Yellow   | CP2WT2 Signal                     |
|    | 2 | 3 | Upper Red      |                                   |
| A  | 3 | 4 | Lower Yellow   |                                   |
|    | 4 | 5 | Lower Red      /                                   |
|    | 5 | 6 |                                                    |
|    | 6 | 7 |                                                    |
|    | 7 | 8 |                                                    |
-------------------------------------------------------------------
|    | 0 | 9 | Normal   \  Switch 2a                              |
|    | 1 |10 | Reverse  /  motor                                  |
|    | 2 |11 | Normal   \  Switch 2b                              |
| B  | 3 |12 | Reverse  /  motor                                  |
|    | 4 |13 |                                                    |
|    | 5 |14 |                                                    |
|    | 6 |15 |                                                    |
```

```
|     | 7 |16 |                                                       |
-------------------------------------------------------------------
|     | 0 |17 |                                                       |
|     | 1 |18 |                                                       |
|     | 2 |19 |                                                       |
| C   | 3 |20 |                                                       |
|     | 4 |21 |                                                       |
|     | 5 |22 |                                                       |
|     | 6 |23 |                                                       |
|     | 7 |24 |                                                       |
-------------------------------------------------------------------
|     | 0 |25 |                                                       |
|     | 1 |26 |                                                       |
|     | 2 |27 |                                                       |
| D   | 3 |28 |                                                       |
|     | 4 |29 |                                                       |
|     | 5 |30 |                                                       |
|     | 6 |31 |                                                       |
|     | 7 |32 |                                                       |
-------------------------------------------------------------------
EOF
```

20.4 Example 4: From Chapter 9 of C/MRI User's Manual V3.0

This example, shown below, implements the yard example from Chapter 9 of C/MRI User's Manual V3.0. [3]

This example uses a single SMINI board. The physical push buttons are replaced by "virtual" push buttons on the computer screen. Otherwise, this code is a drop-in replacement, in Tcl under Linux, for the Quick BASIC code under MS-Windows included in Bruce Chubb's manual.

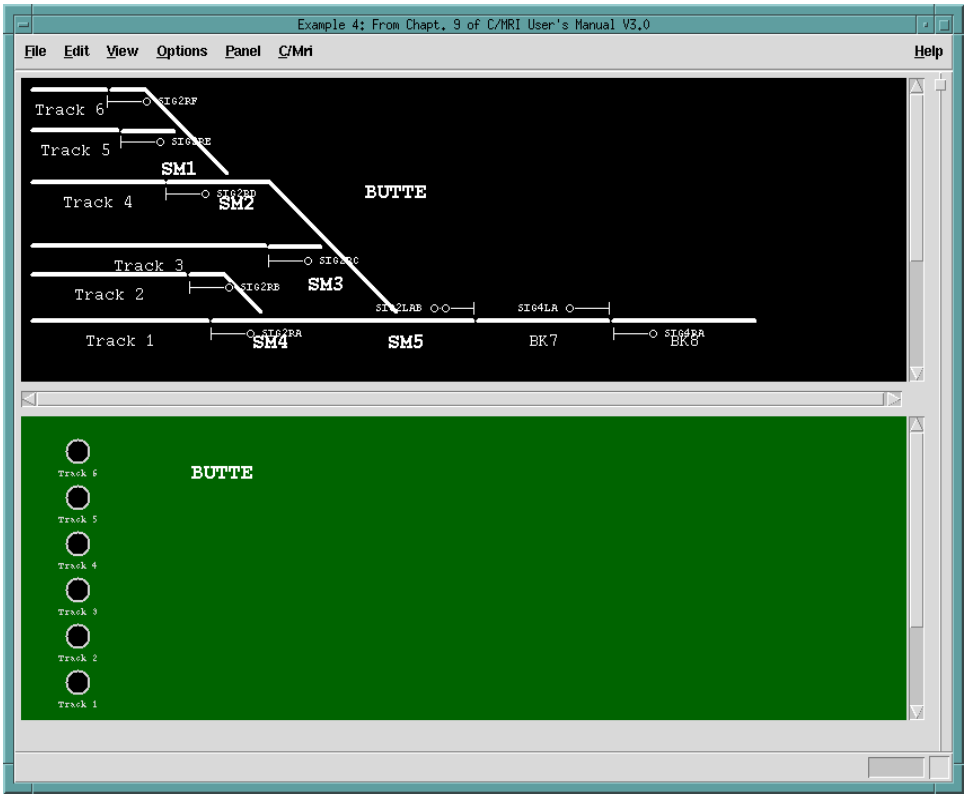


Figure 20.4 Example 4: From Chapter 9 of C/MRI User's Manual V3.0

Here is the code:

```
# Add User code after this line
# From chapter 9 of Computer/Model Railroad Interface User's Manual V3.0.
# See example4.iow for I/O connections for this example.
namespace eval Groups {
  # Radio groups (from push buttons)
  snit::type Group {
    option -buttonmap -readonly yes -default {}
    variable value
    constructor {args} {
      $self configurelist $args
      $self setvalue {}
    }
    method getvalue {} {return $value}
    method setvalue {newvalue} {
      set value $newvalue
      foreach {b v} $options(-buttonmap) {
        if {[string equal "$v" "$value"]} {
          MainWindow ctcpanel seti $b I on
        } else {
          MainWindow ctcpanel seti $b I off
        }
      }
    }
  }
  Group TrackSelect -buttonmap {PB1 Track1 PB2 Track2 PB3 Track3 PB4 Track4
                                PB5 Track5 PB6 Track6}
}
namespace eval Signals {
  # Signal types.  Encapsulates a signal's aspect.
  # Descrete Led types, three aspect (Red, Yellow, Green)
  # See: Fig 3-5 of The Computer / Model Railroad Interface User's Manual V3.0
  snit::type OneHead {
    # Single head signals have four states: dark, green, yellow, or red.
    typevariable aspects -array {
      Dark      0x00
      Green     0x01
      Yellow    0x02
      Red       0x04
    }
    option -signal -default {}
    variable aspectbits
    constructor {args} {
      $self configurelist $args
      set aspectbits $aspects(Dark)
      if {[string length $options(-signal)] > 0} {
        MainWindow ctcpanel setv $options(-signal) dark
      }
    }
    method setaspect {a} {
      set aspectbits $aspects($a)
      if {[string length $options(-signal)] > 0} {
        switch $a {
          Dark {MainWindow ctcpanel setv $options(-signal) dark}
          Green {MainWindow ctcpanel setv $options(-signal) green}
          Yellow {MainWindow ctcpanel setv $options(-signal) yellow}
          Red {MainWindow ctcpanel setv $options(-signal) red}
        }
      }
    }
    method getaspect {}  {return $aspectbits}
  }
  snit::type TwoHead {
    # Two head signals have five states: dark, green over red,
    # yellow over red, red over yellow, and red over red.
    typevariable aspects -array {
      Dark      0x00
      GreenRed  0x11
      YellowRed 0x12
      RedYellow 0x0c
      RedRed    0x14
    }
    option -signal -default {}
    variable aspectbits
    constructor {args} {
      $self configurelist $args
      set aspectbits $aspects(Dark)
      if {[string length $options(-signal)] > 0} {
        MainWindow ctcpanel setv $options(-signal) dark
      }
    }
```

```
    method setaspect {a} {
      set aspectbits $aspects($a)
      if {[string length $options(-signal)] > 0} {
        switch $a {
          Dark {MainWindow ctcpanel setv $options(-signal) dark}
          GreenRed {MainWindow ctcpanel setv $options(-signal) {green red}}
          YellowRed {MainWindow ctcpanel setv $options(-signal) {yellow red}}
          RedYellow {MainWindow ctcpanel setv $options(-signal) {red yellow}}
          RedRed {MainWindow ctcpanel setv $options(-signal) {red red}}
        }
      }
    }
    method getaspect {}  {return $aspectbits}
  }
  snit::type ThreeHead {
    # Three head signals have six states: dark, green over red over red,
    # yellow over red over red, red over yellow over red, red over red
    # over yellow, and red over red over red.
    typevariable aspects -array {
      Dark              0x00
      GreenRedRed       0x51
      YellowRedRed      0x52
      RedYellowRed      0x4c
      RedRedYellow      0x34
      RedRedRed         0x54
    }
    option -signal -default {}
    variable aspectbits
    constructor {args} {
      $self configurelist $args
      set aspectbits $aspects(Dark)
      if {[string length $options(-signal)] > 0} {
        MainWindow ctcpanel setv $options(-signal) dark
      }
    }
    method setaspect {a} {
      set aspectbits $aspects($a)
      if {[string length $options(-signal)] > 0} {
        switch $a {
          Dark {MainWindow ctcpanel setv $options(-signal) dark}
          GreenRedRed {MainWindow ctcpanel setv $options(-signal) {green red red}}
          YellowRedRed {MainWindow ctcpanel setv $options(-signal) {yellow red red}}
          RedYellowRed {MainWindow ctcpanel setv $options(-signal) {red yellow red}}
          RedRedYellow {MainWindow ctcpanel setv $options(-signal) {red red yellow}}
          RedRedRed {MainWindow ctcpanel setv $options(-signal) {red red red}}
        }
      }
    }
    method getaspect {}  {return $aspectbits}
  }
  OneHead SIG2RF -signal SIG2RF
  OneHead SIG2RE -signal SIG2RE
  OneHead SIG2RD -signal SIG2RD
  OneHead SIG2RC -signal SIG2RC
  OneHead SIG2RB -signal SIG2RB
  OneHead SIG2RA -signal SIG2RA
  TwoHead SIG2LAB -signal SIG2LAB
  OneHead SIG4LA -signal SIG4LA
  OneHead SIG4RA -signal SIG4RA
}
namespace eval Blocks {
  # Block type (general trackwork).
  # Encapsulates block occupation detectors.
  #
  snit::type Block {
    # Occupation state values
    typevariable OCC 1
    typevariable CLR 0
    # Occupation state bit
    variable occupiedbit
    constructor {} {
      set occupiedbit $CLR;#    Initialize to clear.
    }
    # Occupation state methods
    method occupiedp {} {return [expr {$occupiedbit == $OCC}]}
    method setoccupied {value} {
      set occupiedbit $value
    }
  }
  Block OS1;# Shared by all switches
  Block BK1
```

```
  Block BK2
  Block BK3
  Block BK4
  Block BK5
  Block BK6
  Block BK7
  Block BK8
}
namespace eval Switches {
  # Switch type (turnout)
  # Encapsulates a switch (turnout), including its OS (delegated to a Block
  # object), its switch motor, and its point position sensor (its state).
  snit::type Switch {
    component block;#                   OS section
    delegate method * to block;#        Delegate block methods
    variable state unknown;#            Sense state (point position)
    # Motor bit values
    typevariable NOR 1;# 01
    typevariable REV 2;# 10
    variable motor;#                    Motor bits -- used to drive switch
#                                       motor.
    constructor {} {
      #                                 Install OS section
      install block using Blocks::Block %AUTO%
      # Initialize motor bits
      set motor $NOR
    }
    # State methods
    method getstate {} {return $state}
    method setstate {statebits} {
      if {$statebits == $NOR} {
        set state normal
      } elseif {$statebits == $REV} {
        set state reverse
      } else {
        set state unknown
      }
    }
    # Motor bit methods
    method motorbits {} {return $motor}
    method setmotor {mv} {
      switch -exact $mv {
        normal {set motor $NOR}
        reverse {set motor $REV}
      }
    }
  }
  Switch SM1
  Switch SM2
  Switch SM3
  Switch SM4
  Switch SM5
}
# Main Loop Start
while {true} {
  # Read all ports
  set Butte_inbits [Butte inputs]
  # Unpack input bits
  set tempbyte [lindex $Butte_inbits 0];# Port A: Blocks 1-7, OS1
  Blocks::BK1 setoccupied [expr {$tempbyte      & 0x01}]
  Blocks::BK2 setoccupied [expr {$tempbyte » 1 & 0x01}]
  Blocks::BK3 setoccupied [expr {$tempbyte » 2 & 0x01}]
  Blocks::BK4 setoccupied [expr {$tempbyte » 3 & 0x01}]
  Blocks::BK5 setoccupied [expr {$tempbyte » 4 & 0x01}]
  Blocks::BK6 setoccupied [expr {$tempbyte » 5 & 0x01}]
  Blocks::OS1 setoccupied [expr {$tempbyte » 6 & 0x01}]
  Blocks::BK7 setoccupied [expr {$tempbyte » 7 & 0x01}]
  set tempbyte [lindex $Butte_inbits 1];# Port B: Block 8
  Blocks::BK8 setoccupied [expr {$tempbyte      & 0x01}]
  # Invoke all trackwork and get occupicency
  MainWindow ctcpanel invoke  OS1a
  MainWindow ctcpanel invoke  OS1b
  MainWindow ctcpanel invoke  OS1c
  MainWindow ctcpanel invoke  OS1d
  MainWindow ctcpanel invoke  OS1e
  MainWindow ctcpanel invoke  OS1f
  MainWindow ctcpanel invoke  OS1g
  MainWindow ctcpanel invoke  OS1h
  MainWindow ctcpanel invoke  OS1i
  MainWindow ctcpanel invoke  OS1j
  MainWindow ctcpanel invoke  OS1k
```

```
MainWindow ctcpanel invoke  OS1l
MainWindow ctcpanel invoke  OS1m
MainWindow ctcpanel invoke  OS1n
MainWindow ctcpanel invoke  BK1
MainWindow ctcpanel invoke  BK2
MainWindow ctcpanel invoke  BK3
MainWindow ctcpanel invoke  BK4
MainWindow ctcpanel invoke  BK5
MainWindow ctcpanel invoke  BK6
MainWindow ctcpanel invoke  BK7
MainWindow ctcpanel invoke  BK8
# Process switch machines
if {![Blocks::OS1 occupiedp]} {
  switch [::Groups::TrackSelect getvalue] {
    Track1 {Switches::SM4 setmotor normal;Switches::SM5 setmotor normal}
    Track2 {Switches::SM4 setmotor reverse;Switches::SM5 setmotor normal}
    Track3 {Switches::SM5 setmotor reverse;Switches::SM3 setmotor reverse}
    Track4 {Switches::SM5 setmotor reverse;Switches::SM3 setmotor normal;
            Switches::SM2 setmotor normal}
    Track5 {Switches::SM5 setmotor reverse;Switches::SM3 setmotor normal;
            Switches::SM2 setmotor reverse;Switches::SM1 setmotor reverse}
    Track6 {Switches::SM5 setmotor reverse;Switches::SM3 setmotor normal;
            Switches::SM2 setmotor reverse;Switches::SM1 setmotor normal}
  }
}
MainWindow ctcpanel invoke  SM1
MainWindow ctcpanel invoke  SM2
MainWindow ctcpanel invoke  SM3
MainWindow ctcpanel invoke  SM4
MainWindow ctcpanel invoke  SM5
# Compute direction of traffic
if {[Block::OS1 occupiedp]} {
  set dot1 EastBound
} else {
  set dot1 Westbound
}
# Calculate SIG4RA
Signals::SIG4RA setaspect Red
if {$dot1 ne WestBound && ![Block::BK8 occupiedp]} {
  Signals::SIG4RA setaspect Green
}
# Calculate Butte exit signals
Signals::SIG2RA setaspect Red
Signals::SIG2RB setaspect Red
Signals::SIG2RC setaspect Red
Signals::SIG2RD setaspect Red
Signals::SIG2RE setaspect Red
Signals::SIG2RF setaspect Red
set ExitSig Red
if {$dot1 ne WestBound && ![Blocks::OS1 occupiedp] && ![Blocks::BK7 occupiedp]} {
  set ExitSig Yellow
  if {[MainWindow ctcpanel getv SIG4RA] ne red} {set ExitSig Green}
  switch [::Groups::TrackSelect getvalue] {
    Track1 {Signals::SIG2RA setaspect $ExitSig}
    Track2 {Signals::SIG2RB setaspect $ExitSig}
    Track3 {Signals::SIG2RC setaspect $ExitSig}
    Track4 {Signals::SIG2RD setaspect $ExitSig}
    Track5 {Signals::SIG2RE setaspect $ExitSig}
    Track6 {Signals::SIG2RF setaspect $ExitSig}
  }
}
Signals::SIG2LAB setaspect RedRed
if {![Blocks::OS1 occupiedp]} {
  switch [::Groups::TrackSelect getvalue] {
    Track1 {if {![Blocks::BK1 occupiedp]} {Signals::SIG2LAB setaspect YellowRed}}
    Track2 {if {![Blocks::BK2 occupiedp]} {Signals::SIG2LAB setaspect RedYellow}}
    Track3 {if {![Blocks::BK3 occupiedp]} {Signals::SIG2LAB setaspect RedYellow}}
    Track4 {if {![Blocks::BK4 occupiedp]} {Signals::SIG2LAB setaspect RedYellow}}
    Track5 {if {![Blocks::BK5 occupiedp]} {Signals::SIG2LAB setaspect RedYellow}}
    Track6 {if {![Blocks::BK5 occupiedp]} {Signals::SIG2LAB setaspect RedYellow}}
  }
}
Signals::SIG4LA setaspect Red
if {$dot1 ne EastBound && ![Blocks::BK7 occupiedp]} {
  Signals::SIG4LA setaspect Yellow
  if {[MainWindow ctcpanel getv SIG2LAB] ne {red red}} {
    Signals::SIG4LA setaspect Green
  }
}
# Pack output bits
# Port A, Card 0:
```

```
  set     Butte_outbits  [expr {[Signals::SIG2LAB getaspect] | \
                                [Signals::SIG2RA  getaspect] « 5}]
  # Port B, Card 0:
  lappend Butte_outbits  [expr {[Signals::SIG2RB  getaspect] | \
                                [Signals::SIG2RC  getaspect] « 3 | \
                                [Switches::SM1    motorbits] « 6}]
  # Port C, Card 0:
  lappend Butte_outbits  [expr {[Signals::SIG2RD  getaspect] | \
                                [Signals::SIG2RE  getaspect] « 3 | \
                                [Switches::SM2    motorbits] « 6}]
  # Port A, Card 2:
  lappend Butte_outbits  [expr {[Signals::SIG2RF  getaspect] | \
                                [Switches::SM3    motorbits] « 3 | \
                                [Switches::SM4    motorbits] « 5}]
  # Port B, Card 2:
  lappend Butte_outbits  [expr {[Signals::SIG4LA  getaspect] | \
                                [Signals::SIG4RA  getaspect] « 3 | \
                                [Switches::SM5    motorbits] « 6}]
  # Port C, Card 2:
  lappend Butte_outbits 0
  # Write all output ports
  eval [list Butte outputs] $Butte_outbits
  update;# Update display
}
# Main Loop End
```

And the I/O Worksheet:

```
SMINI @ UA 0 (Butte)
Card No. 0 Output
-----------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                |
-----------------------------------------------------------------
|    | 0 | 1 | Upper Green    \                               |
|    | 1 | 2 | Upper Yellow   | SIG2LAB Signal                |
|    | 2 | 3 | Upper Red      |                               |
| A  | 3 | 4 | Lower Yellow   |                               |
|    | 4 | 5 | Lower Red      /                               |
|    | 5 | 6 | Green          \                               |
|    | 6 | 7 | Yellow         | SIG2RA                        |
|    | 7 | 8 | Red            /                               |
-----------------------------------------------------------------
|    | 0 | 9 | Green          \                               |
|    | 1 |10 | Yellow         | SIG2RB Signal                 |
|    | 2 |11 | Red            /                               |
| B  | 3 |12 | Green          \                               |
|    | 4 |13 | Yellow         | SIG2RC Signal                 |
|    | 5 |14 | Red            /                               |
|    | 6 |15 | Normal         \ SM1                           |
|    | 7 |16 | Reverse        /                               |
-----------------------------------------------------------------
|    | 0 |17 | Green          \                               |
|    | 1 |18 | Yellow         | SIG2RD Signal                 |
|    | 2 |19 | Red            /                               |
| C  | 3 |20 | Green          \                               |
|    | 4 |21 | Yellow         | SIG2RE Signal                 |
|    | 5 |22 | Red            /                               |
|    | 6 |23 | Normal         \ SM2                           |
|    | 7 |24 | Reverse        /                               |
-----------------------------------------------------------------
|    | 0 |25 |                                                |
|    | 1 |26 |                                                |
|    | 2 |27 |                                                |
| D  | 3 |28 |                                                |
|    | 4 |29 |                                                |
|    | 5 |30 |                                                |
|    | 6 |31 |                                                |
|    | 7 |32 |                                                |
-----------------------------------------------------------------
Card No. 1 Input
-----------------------------------------------------------------
|PORT|BIT|PIN|DESCRIPTION OF FUNCTION PERFORMED                |
-----------------------------------------------------------------
|    | 0 | 1 | BK1 Occupation Detector                        |
|    | 1 | 2 | BK2 Occupation Detector                        |
|    | 2 | 3 | BK3 Occupation Detector                        |
| A  | 3 | 4 | BK4 Occupation Detector                        |
|    | 4 | 5 | BK5 Occupation Detector                        |
|    | 5 | 6 | BK6 Occupation Detector                        |
|    | 6 | 7 | OS1 Occupation Detector                        |
|    | 7 | 8 | BK7 Occupation Detector                        |
```

	0	9	BK8 Occupation Detector	
	1	10		
	2	11		
B	3	12		
	4	13		
	5	14		
	6	15		
	7	16		
	0	17		
	1	18		
	2	19		
C	3	20		
	4	21		
	5	22		
	6	23		
	7	24		
	0	25		
	1	26		
	2	27		
D	3	28		
	4	29		
	5	30		
	6	31		
	7	32		

Card No. 2 Output

PORT	BIT	PIN	DESCRIPTION OF FUNCTION PERFORMED	
	0	1	Green	\
	1	2	Yellow	| SIG2RF Signal
	2	3	Red	/
A	3	4	Normal	\ SM3
	4	5	Reverse	/
	5	6	Normal	\ SM4
	6	7	Reverse	/
	7	8		
	0	9	Green	\
	1	10	Yellow	| SIG4LA Signal
	2	11	Red	/
B	3	12	Green	\
	4	13	Yellow	| SIG4RA Signal
	5	14	Red	/
	6	15	Normal	\ SM5
	7	16	Reverse	/
	0	17		
	1	18		
	2	19		
C	3	20		
	4	21		
	5	22		
	6	23		
	7	24		
	0	25		
	1	26		
	2	27		
D	3	28		
	4	29		
	5	30		
	6	31		
	7	32		

Chapter 21

SatelliteDaemon

Satellite computer Daemon

21.1 SYNOPSIS

```
SatelliteDaemon [-debug] [-port listenport] [-log logfilename] [-nodaemon] [-addpath path] [-addpackage path]
SatelliteDaemon -extenddaemon newDaemon [-tclkit tclkit] packageDir ...
```

21.2 DESCRIPTION

This is the deamon program that runs on 'satellite' computers on a network of computers controlling a layout. These satellite computers could be Raspberry Pis (running Rasperian) or small Intel/AMD systems (eg mini-Itx systems or other Intel/AMD SBCs running some version of Linux). Somewhere on the network would be a (desktop) computer running the master (client) package, typically in the context of a Dispatcher control panel.
The satellite systems would be connected to a collection of USB interface boards (or possibly using the GPIO pins on a Raspberry) that would interface to various actuating devices on the layout, such as turnout motors, signals, uncoupling magnets, as well as various sensors that might be in use.

The second form (SatelliteDaemon -extenddaemon newDaemon ...) of the command creates a new StarPak with the listed package dirs copied into the StarPak. The -tclkit defaults to /usr/local/bin/tclkit.

21.3 OPTIONS

- -debug This option turns on verbose debug logging.
- -port This option selects the port to listen on (default: 40000).
- -log This option selects the name of the log filename (default: $HOME/SatelliteDaemon.log).
- -nodaemon This option supresses daemonisation to facilitate debugging.
- -addpath This option adds additional search paths for packages. (These packages are always available: the core Tcl packages, snit, Azatrax, and Cmri.)
- -addpackage This option loads additional packages into the daemon's internal virtual file system.
- -extenddaemon This option creates a new version of the daemon with additional package directories included.

21.4 PROTOCOL

The protocol is simple. The daemon creates a 'safe' interpreter and simply feeds the command stream coming in to this interpreter and sends this interpreter's output back to the master.

21.5 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 22

raildriverd

Raildriver USB Hotplug Daemon

22.1 SYNOPSIS

```
raildriverd [-debug] busnum devnum
```

22.2 DESCRIPTION

This is the deamon program for the Rail Driver. It is started by the USB Hotplug code. See Hotplugging scripts and setup. for details. It should not be started or stopped by hand!

The API to use this deamon is described in *Model Railroad System Programming Guides*, Part I. User programs connect to this deamon through a Tcp/Ip port. It allows multiple programs to access a single Raildriver device. These programs can then in turn implement various functionallity for the various levers, knobs, switches, and buttons on the Raildriver device.

22.3 OPTIONS

- -debug This option turns on verbose debug logging.

22.4 PARAMETERS

- busnum This is the USB bus number the device is connected to.
- devnum This is the USB device number the device is connected to.

22.5 Hotplugging scripts and setup.

There are two ways to set up auto starting of this daemon.

1. Using the Hotplug daemon. Copy the raildriverd.hotplug script to /etc/hotplug/usb/ as raildriverd Use the print-usb-usermap stript to append a line to /etc/hotplug/usb.usermap.
2. Using udev. Copy 90-raildriver.rules to /etc/udev/rules.d/ and copy raildriverd.udev to /lib/udev/ as raildriverd

22.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 23

OpenLCB Tcp/Ip Hub Server

OpenLCB Tcp Hub daemon.

23.1 SYNOPSIS

OpenLCBTcpHub [-host localhost] [-port 12000] [-debug]

23.2 DESCRIPTION

This program is a server daemon that implements a hub for Native OpenLCB over Tcp/Ip that accepts connections from OpenLCB over Tcp/Ip nodes and forwards OpenLCB messages between clients.

23.3 PARAMETERS

none

23.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCBTcpHub.log
- -host hostname The name or IP address of the host to bind to. Defaults to localhost (binds only to the local loopback device). Using an address of 0.0.0.0 will bind to all interfaces.
- -port portnumber The Tcp/Ip port to listen on. Defaults to 12000.
- -debug Turns on debug logging.
- -remote host[:port], -remote0 host[:port], -remote1 host[:port], ... -remote9 host[:port] Optional remote Tcp/Ip hubs.

23.5 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 24

OpenLCB GridConnect Tcp/Ip Hub Server

OpenLCB GridConnect Tcp Hub daemon.

24.1 SYNOPSIS

OpenLCBGCTcpHub [-host localhost] [-port 12021] [-debug] [connection options]

24.2 DESCRIPTION

This program is a server daemon that implements a hub for OpenLCB over Tcp/Ip that accepts connections from Open↩ LCB over GridConnect over Tcp/Ip nodes and forwards OpenLCB messages between clients. It can also connect to physical CAN busses using GridConnect messaging over (USB) Serial port connection.

24.3 PARAMETERS

none

24.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCBGCTcpHub.log
- -host hostname The name or IP address of the host to bind to. Defaults to localhost (binds only to the local loopback device). Using an address of 0.0.0.0 will bind to all interfaces.
- -port portnumber The Tcp/Ip port to listen on. Defaults to 12021.
- -debug Turns on debug logging.

- -dev ttydev, -dev0 ttydev, -dev1 ttydev, ... -dev9 ttydev Optional serial ports connected to CAN busses using GridConnect.
- -remote host[:port], -remote0 host[:port], -remote1 host[:port], ... -remote9 host[:port] Optional remote Tcp/Ip hubs using GridConnect.
- -can socketname, -can0 socketname, -can1 socketname, ... -can9 socketname Optional CAN Socket connected CAN networks.

24.5 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 25

OpenLCB Router Daemon (Server)

Routes between OpenLCB GridConnect/CAN and binary OpenLCB over Tcp/Ip

25.1 SYNOPSIS

Router [-bhost localhost] [-bport 12000] [-cmode Tcpip|Socket|USB] [-chost localhost] [-cport 12021] [-csocket can0] [-cdevice /dev/ttyACM0] [-nid 05:01:01:01:22:00] [-log Router.log] [-debug] [-nodename ""] [-nodedescription ""]

25.2 DESCRIPTION

This program is a server daemon that implements a router between an OpenLCB/CAN segment and a native OpenLCB over Tcp/Ip network.

25.3 PARAMETERS

none

25.4 OPTIONS

- -bhost The binary OpenLCB over Tcp/Ip host to connect to.
- -bport The tcp port to connect with.
- -cmode The CAN If mode: Tcpip means GridConnect over Tcp/Ip, Socket means use a CAN famile Socket (Linux only) (using the TclSocketCAN API), and USB means using a USB Serial port connection using GridConnect (such as a RR-CirKits USB Buffer-LCC).
- -chost The GridConnect over Tcp/Ip host to connect to (only when -cmode is Tcpip).

- -cport The tcp port to connect with (only when -cmode is Tcpip).
- -csocket The CAN socket name (only when -cmode is Socket).
- -cdevice The tty device to connect to (only when -cmode is USB).
- -nid The OpenLCB Node ID for the router.
- -log The file to use for logging.
- -debug Enable debugging messages.
- -nodename The name of this router node.
- -nodedescription The description of this router node.

25.5 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 26

OpenLCB MRD2 Node

OpenLCB MRD2 node

26.1 SYNOPSIS

OpenLCB_MRD2 [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

26.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for one or more Azatrax MRD2 devices.

26.3 PARAMETERS

none

26.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_MRD2.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is mrd2conf.xml.
- -debug Turns on debug logging.

26.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

26.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 27

OpenLCB PiGPIO node

OpenLCB PiGPIO node

27.1 SYNOPSIS

OpenLCB_PiGPIO [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

27.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for the GPIO pins on a Raspberry Pi.

27.3 PARAMETERS

none

27.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_PiGPIO.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is pigpioconf.xml.
- -debug Turns on debug logging.

27.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

27.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 28

OpenLCB PiMCP23008 node

OpenLCB PiMCP23008 node

28.1 SYNOPSIS

OpenLCB_PiMCP23008 [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

28.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for the GPIO pins provided by a MCP23008 I2C port expander on a Raspberry Pi.

28.3 PARAMETERS

None

28.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_PiMCP23008.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is pimcp23008conf.xml.
- -debug Turns on debug logging.

28.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

28.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 29

OpenLCB PiMCP23017 node

OpenLCB PiMCP23017 node

29.1 SYNOPSIS

OpenLCB_PiMCP23017 [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

29.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for the GPIO pins provided by a MCP23017 I2C port expander on a Raspberry Pi.

29.3 PARAMETERS

None

29.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_PiMCP23017.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is pimcp23017conf.xml.
- -debug Turns on debug logging.

29.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

29.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 30

OpenLCB PiMCP23017 as signal driver node

OpenLCB PiMCP23017 as signal driver node

30.1 SYNOPSIS

OpenLCB_PiMCP23017_signal [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

30.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for the GPIO pins provided by a MCP23017 I2C port expander on a Raspberry Pi. This version groups the pins as signal heads. All pins are set as outputs.

30.3 PARAMETERS

None

30.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_PiMCP23017_signal.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is pimcp23017signalconf.xml.
- -debug Turns on debug logging.

30.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

30.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 31

OpenLCB program for the MCP23017-based quad signal head HAT

OpenLCB OpenLCB for the MCP23017-based quad signal head HAT

31.1 SYNOPSIS

OpenLCB_QuadSignal [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

31.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for the MCP23017-based quad signal head HAT for the Raspberry Pi. Each signal mast can have 1, 2, or 3 "heads". Each head has four "lamps" (unused lamps can be set to "None"). For a given aspect, a lamp can be on, off, blink, or reverse blink.

31.3 PARAMETERS

None

31.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_QuadSignal.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is quadsignalconf.xml.
- -debug Turns on debug logging.

31.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

31.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 32

OpenLCB PiSPIMax7221 node

OpenLCB PiSPIMax7221 node

32.1 SYNOPSIS

OpenLCB_PiSPIMax7221 [-configure] [-sampleconfiguration] [-debug] [-test all|m-n] [-configuration confgile]

32.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for the a Max7221 based signal driver.

32.3 PARAMETERS

None

32.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_PiSPIMax7221.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is PiSPIMax7221conf.xml.
- -debug Turns on debug logging.
- -test all|n-m Test all or signals n though m. Run continously until killed.

32.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

32.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 33

OpenLCB Virtual Track Circuits node

OpenLCB Virtual Track Circuits node

33.1 SYNOPSIS

OpenLCB_TrackCircuits [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

33.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for one or more Virtual Track Circuits (much like the track circuits coded in the RR Cirkits Tower-LCC nodes).

There are seven (7) regular aspect events (**Clear**, **Advance Approach**, **Approach Limited**, **Approach Medium**, **Approach**, **Approach Slow**, and **Accelerated Tumble Down**), plus **Start**, **Non-Vital** (**occupied**), **Non-Vital** (**normal**), **Power**/**Lamp** (**failed**), and **Power**/**Lamp** (**normal**).

33.2.1 Code rate and aspect.

- 7 Clear.
- 4 Advance Approach.
- 3 Approach Limited.
- 8 Approach Medium.
- 2 Approach.
- 9 Approach Slow.
- 6 Accelerated Tumble Down.
- 5 Non-Vital code indicating track occpancy, or a hand-thrown switch in the block out of normal correspondence.
- M Non-Vital code indicating power off in the block, or a lamp out of condition in the block. Power Off will indicate from east end CP, lamp out from the west end CP.

33.3 PARAMETERS

none

33.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_TrackCircuits.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is tracksconf.xml.
- -debug Turns on debug logging.

33.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

33.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 34

OpenLCB Logic node

OpenLCB Logic node

34.1 SYNOPSIS

OpenLCB_Logic [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

34.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for one or more Logic Element (much like the logic elements coded in the RR Cirkits Tower-LCC nodes).

Each logic element has two variable inputs, variable 1 and variable 2, which can be set to true or false using LCC events. There are seven (7) boolean operators: **and**, **or**, **xor**, **and change**, **or change**, **variable 1 then variable 2**, and **constant true**. Logic elements can be either single, or part of a group. There are two group types, mast and ladder. A mast group is always evaluated from top to bottom and terminates at the first true result which produces an action. A ladder group is evaluated from the triggered logic to the bottom and all true results result in actions. Actions consist of up to four events being produced, either right away or after a delay. The actions can be retriggerable or not.

34.3 PARAMETERS

none

34.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_Logic.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is logicconf.xml.
- -debug Turns on debug logging.

34.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

34.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 35

OpenLCB Acela Node

OpenLCB Acela node

35.1 SYNOPSIS

OpenLCB_Acela [-configure] [-sampleconfiguration] [-debug] [-configuration configfile]

35.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for an Acela network.

35.3 PARAMETERS

none

35.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_Acela.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration configfile Sets the name of the configuration (XML) file. The default is acelaconf.xml.
- -debug Turns on debug logging.

35.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

35.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 36

OpenLCB CMRI Node

OpenLCB CMRI node

36.1 SYNOPSIS

OpenLCB_CMRI [-configure] [-sampleconfiguration] [-debug] [-configuration confgile]

36.2 DESCRIPTION

This program is a daemon that implements an OpenLCB node for one or more CMRI nodes on a CMRI network.

36.3 PARAMETERS

none

36.4 OPTIONS

- -log logfilename The name of the logfile. Defaults to OpenLCB_CMRI.log
- -configure Enter an interactive GUI configuration tool. This tool creates or edits an XML configuration file.
- -sampleconfiguration Creates a **sample** configuration file that can then be hand edited (with a handy text editor like emacs or vim).
- -configuration confgile Sets the name of the configuration (XML) file. The default is cmriconf.xml.
- -debug Turns on debug logging.

36.5 CONFIGURATION

The configuration file for this program is an XML formatted file. Please refer to the OpenLCB Daemons (Hubs and Virtual nodes) chapter of the User Manual for the details on the schema for this XML formatted file. Also note that this program contains a built-in editor for its own configuration file.

36.6 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 37

JMRI Tables to LayoutDB converter

Converts a JMRI Table file to a LayoutDB file

37.1 SYNOPSIS

JMRITable2LayoutDB jmrixml layoutdbxml

37.2 DESCRIPTION

Convert a JMRI Table XML file to a LayoutControlDB xml file.

37.3 PARAMETERS

- jmrixml The JMRI XML file to convert.
- layoutdbxml The LayoutControlDB xml file to output

37.4 OPTIONS

None.

37.5 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 38

LayoutDB to JMRI Tables converter

Converts a LayoutDB file to a JMRI Table file

38.1 SYNOPSIS

LayoutDB2JMRITable layoutdbxml jmrixml

38.2 DESCRIPTION

Convert a Layout Control DB XML file to a JMRI Table XML file

38.3 PARAMETERS

- layoutdbxml The Layout Control DB XML file to convert
- jmrixml The JMRI Table XML file to output

38.4 OPTIONS

None

38.5 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 39

Offline LCC Node Editor

An offline node configuration editor. Edits a backup config file for a LCC node.

39.1 SYNOPSIS

OfflineLCCNodeEditor xmlfile [backupconfigfile ...]

39.2 DESCRIPTION

Uses the saved CDI XML file from the node to create the configuration editor display and loads a backup config file into the configuration editor and saves changes to a new backup config file.

39.3 PARAMETERS

- xmlfile - the CDI XML file for the type of node. Required, no default.
- backupconfigfile - [optional] backup config file(s) to edit.

39.4 OPTIONS

39.4.1 X11 Resource Options

- -colormap: Colormap for main window
- -display: Display to use
- -geometry: Initial geometry for window
- -name: Name to use for application
- -sync: Use synchronous mode for display server
- -visual: Visual for main window
- -use: Id of window in which to embed application

39.4.2 Other options

- -help Print a short help message and exit.
- -debug Turn on debug output.

39.5 AUTHOR

Robert Heller <heller@deepsoft.com>

Chapter 40

Help

This help window contains some basic navigation features. There are buttons for traversing the history stack. There are also key bindings within the help window itself:

- **s** Search forward. Searches forward in the text for the next occurrence of the specified text.
- **r** Search backward. Searches backward in the text for the next occurrence of the specified text.
- **f** History forward. Goes to the next page in the history stack.
- **b** History backward. Goes to the previous page in the history stack.
- **Tab** Next link. Goes to the next hyperlink.
- **Control-Tab** Previous link. Goes to the previous hyperlink.

Chapter 41

Version

TT Support Library version is 1.0.2. FCF Support Library version is 1.0.4. CMri Library version is 1.0.0. System patch level is 2.2.1. Azatrax Library version is 1.0.0.

Chapter 42

GNU GENERAL PUBLIC LICENSE

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

1. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

```
a) You must cause the modified files to carry prominent notices
stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in
whole or in part contains or is derived from the Program or any
part thereof, to be licensed as a whole at no charge to all third
parties under the terms of this License.

c) If the modified program normally reads commands interactively
when run, you must cause it, when started running for such
interactive use in the most ordinary way, to print or display an
announcement including an appropriate copyright notice and a
notice that there is no warranty (or else, saying that you provide
a warranty) and that users may redistribute the program under
these conditions, and telling the user how to view a copy of this
License.  (Exception: if the Program itself is interactive but
does not normally print such an announcement, your work based on
the Program is not required to print an announcement.)
```

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

1. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

```
a) Accompany it with the complete corresponding machine-readable
source code, which must be distributed under the terms of Sections
1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three
years, to give any third party, for a charge no more than your
cost of physically performing source distribution, a complete
machine-readable copy of the corresponding source code, to be
distributed under the terms of Sections 1 and 2 above on a medium
customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer
to distribute corresponding source code.  (This alternative is
allowed only for noncommercial distribution and only if you
received the program in object code or executable form with such
an offer, in accord with Subsection b above.)
```

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

1. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

1. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

    (a) If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

1. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

1. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

1. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

1. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PR↩ OGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WIT↩ HOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSE↩ QUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
\<one line to give the program's name and a brief idea of what it does.\>
Copyright (C) \<year\>  \<name of author\>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year  name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Bibliography

[1] Bruce Chubb. *How to Operate Your Model Railroad*. Kalmbach Books, 1977, 1978, 1991.

[2] Bruce Chubb. *Build Your Own Universal Computer Interface*. Tab Books, 1989.

[3] Bruce Chubb. *The Computer/Model Railroad Interfase (C/MRI) User's Manual*, 2003.

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

Index

cabs, creating, 56, 64, 74
CAN, 9
cars, saving, 99

Freight Car Forwarder, main GUI, 93

LCC, 9
LM100, 49
LM50, 49

MRD2-S, 43, 44
MRD2-U, 43, 44

notes, creating and editing, 75

Ohm's Law, 133
OpenLCB, 9

Reports, printing, 122

SL2, 46
SMINI, 5
SR4, 43, 45
station stops, creating, 56, 64
stations, duplicate, linking and unlinking, 72
storage tracks, creating, 56, 64, 73
SUSIC, 5, 6

timetable, printing, 81
train, adding a schedule, 57
train, adding storage tracks, 57, 70
train, deleting, 71
trains, creating, 56, 67

USIC, 5, 6

Yard Lists, printing, 101, 119